



# **Nucleus® Networking Guide**

Release 2013.08

August 2013

---

**© 2007-2013 Mentor Graphics Corporation  
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**U.S. GOVERNMENT LICENSE RIGHTS:** The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

**TRADEMARKS:** The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the owner of the Mark, as applicable. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: [www.mentor.com/trademarks](http://www.mentor.com/trademarks).

Mentor Graphics Corporation  
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777  
Telephone: 503.685.7000  
Toll-Free Telephone: 800.592.2210  
Website: [www.mentor.com](http://www.mentor.com)  
SupportNet: [supportnet.mentor.com/](http://supportnet.mentor.com/)

Send Feedback on Documentation: [supportnet.mentor.com/doc\\_feedback\\_form](http://supportnet.mentor.com/doc_feedback_form)

# Table of Contents

---

## Chapter 1

<b>Nucleus Networking Components</b> .....	<b>51</b>
Build Configurations .....	51
Required Include File. ....	51
Networking Components .....	52

## Chapter 2

<b>NET</b> .....	<b>55</b>
NET Overview .....	55
Net Data Structures .....	56
DHCP6_IA_ADDR_STRUCT .....	57
DEV6_PRFX_ENTRY .....	57
ARP_ENTRY .....	58
NU_BOOTP_STRUCT .....	59
NU_DEVICE .....	60
IP6_POLICY_ENTRY .....	62
DHCP_DUID_STRUCT .....	62
NU_HOSTENT .....	63
SCK_IOCTL_OPTION .....	64
DEV6_RTR_OPTS .....	65
NU_DHCP_STRUCT .....	66
DHCP6_CLIENT .....	68
RIP2_STRUCT .....	68
RIPNG_STRUCT .....	69
NU_DNS_SD_SERVICE .....	70
NU_DNS_SD_INSTANCE .....	71
UM_USER .....	71
UPDATED_ROUTE_NODE .....	72
cmsghdr .....	74
msghdr .....	74
in6_pktinfo .....	75
addr_struct .....	76
ip6_hbh .....	76
ip6_rthdr .....	77
sck_linger_struct .....	78
if_nameindex .....	78
Net API Functions .....	79
DHCP6_Build_IA_NA_Option (IPv6) .....	89
DHCP6_Build_Option_Request_Option (IPv6) .....	91
DHCP6_Build_User_Class_Option (IPv6) .....	93
DHCP6_Build_Vendor_Class_Option (IPv6) .....	95
DHCP6_Build_Vendor_Specific_Info_Option (IPv6) .....	97
NETBOOT_Wait_For_Network_Up .....	99
NU_Abort (IPv4/IPv6) .....	100

---

NU_Accept (IPv4/IPv6) . . . . .	101
NU_Add_DNS_Server (IPv4) . . . . .	104
NU_Add_DNS_Server2 (IPv4/IPv6) . . . . .	106
NU_Add_IP_To_Device (IPv6) . . . . .	108
NU_Add_Prefix_Entry (IPv6) . . . . .	110
NU_Add_Route (IPv4) . . . . .	112
NU_Add_Route6 (IPv6) . . . . .	114
NU_ARP_Update . . . . .	116
NU_Attach_IP_To_Device (IPv4) . . . . .	117
NU_Bind (IPv4/IPv6) . . . . .	119
NU_Bootp (IPv4) . . . . .	122
NU_Clear_Internal_Log . . . . .	126
NU_Close_Socket (IPv4/IPv6) . . . . .	127
NU_CMSG_DATA (IPv6) . . . . .	129
NU_CMSG_FIRSTHDR (IPv6) . . . . .	131
NU_CMSG_LEN (IPv6) . . . . .	133
NU_CMSG_NXTHDR (IPv6) . . . . .	135
NU_CMSG_SPACE (IPv6) . . . . .	137
NU_Configure_IPv6_Interface (IPv6) . . . . .	138
NU_Configure_Policy_Table (IPv6) . . . . .	140
NU_Configure_Router (IPv6) . . . . .	143
NU_Connect (IPv4/IPv6) . . . . .	146
NU_Create_IPv4_Mapped_Addr (IPv6) . . . . .	150
NU_Delete_DNS_Server (IPv4) . . . . .	151
NU_Delete_DNS_Server2 (IPv4/IPv6) . . . . .	152
NU_Delete_Host_Entry (IPv4/IPv6) . . . . .	154
NU_Delete_Prefix_Entry (IPv6) . . . . .	156
NU_Delete_Route (IPv4) . . . . .	157
NU_Delete_Route2 (IPv4/IPv6) . . . . .	158
NU_Detach_IP_From_Device (IPv4/IPv6) . . . . .	160
NU_Device_Up (IPv4/IPv6) . . . . .	161
NU_Dhcp (IPv4) . . . . .	162
NU_Dhcp6 (IPv6) . . . . .	166
NU_Dhcp_Release (IPv4) . . . . .	168
NU_Dhcp6_Shutdown (IPv6) . . . . .	170
NU_DNS_SD_Browse . . . . .	172
NU_DNS_SD_Look_Up . . . . .	174
NU_DNS_SD_Refresh . . . . .	176
NU_DNS_Register_Service . . . . .	178
NU_DNS_Build_Key . . . . .	180
NU_DNS_Parse_Key . . . . .	181
NU_Ethernet_Link_Down . . . . .	183
NU_Ethernet_Link_Up . . . . .	184
NU_Fcntl (IPv4/IPv6) . . . . .	185
NU_FD_Check (IPv4/IPv6) . . . . .	187
NU_FD_Init (IPv4/IPv6) . . . . .	189
NU_FD_Reset (IPv4/IPv6) . . . . .	191
NU_FD_Set (IPv4/IPv6) . . . . .	193
NU_Find_Next_Route (IPv4/IPv6) . . . . .	195



---

NU_Find_Next_Route_Entry . . . . .	196
NU_Find_Route_By_Gateway (IPv4/IPv6) . . . . .	197
NU_Find_Socket (IPv4/IPv6) . . . . .	199
NU_Get_Default_Route (IPv4) . . . . .	201
NU_Free_Host_Entry (IPv4/IPv6) . . . . .	202
NU_Get_Default_TTL (IPv4) . . . . .	203
NU_Get_DHCP_DUID (IPv4/IPv6) . . . . .	204
NU_Get_DHCP_IAID (IPv4/IPv6) . . . . .	205
NU_Get_DNS_Servers (IPv4) . . . . .	207
NU_Get_DNS_Servers2 (IPv4/IPv6) . . . . .	209
NU_Get_Domain_Name (IPv4/IPv6) . . . . .	211
NU_Get_Host_By_Addr (IPv4/IPv6) . . . . .	212
NU_Get_Host_By_Name (IPv4) . . . . .	214
NU_Get_Host_MX . . . . .	216
NU_Get_Host_Name (IPv4/IPv6) . . . . .	218
NU_Get_IP_Forwarding (IPv4/IPv6) . . . . .	219
NU_Get_IP_Node_By_Addr (IPv4/IPv6) . . . . .	220
NU_Get_IP_Node_By_Name (IPv4/IPv6) . . . . .	222
NU_Get_Link_Local_Addr (IPv6) . . . . .	227
NU_Get_Peer_Name (IPv4/IPv6) . . . . .	228
NU_Get_PMTU (IPv4/IPv6) . . . . .	230
NU_Get_Reasm_Max_Size (IPv4) . . . . .	232
NU_Get_Sock_Name (IPv4/IPv6) . . . . .	233
NU_Getsockopt (IPv4/IPv6) . . . . .	235
NU_Getsockopt_IP_BROADCAST_IF (IPv4) . . . . .	244
NU_Getsockopt_IP_HDRINCL (IPv4) . . . . .	246
NU_Getsockopt_IP_MULTICAST_IF (IPv4) . . . . .	248
NU_Getsockopt_IP_MULTICAST_TTL (IPv4) . . . . .	250
NU_Getsockopt_IP_RECVIFADDR (IPv4) . . . . .	252
NU_Getsockopt_IP_TOS (IPv4) . . . . .	254
NU_Getsockopt_IP_TTL (IPv4) . . . . .	256
NU_Getsockopt_IPV6_CHECKSUM (IPv6) . . . . .	258
NU_Getsockopt_IPV6_DSTOPTS (IPv6) . . . . .	260
NU_Getsockopt_IPV6_HOPOPTS (IPv6) . . . . .	262
NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6) . . . . .	264
NU_Getsockopt_IPV6_MULTICAST_IF (IPv6) . . . . .	266
NU_Getsockopt_IPV6_NEXTHOP (IPv6) . . . . .	268
NU_Getsockopt_IPV6_PKTINFO (IPv6) . . . . .	270
NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6) . . . . .	272
NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6) . . . . .	274
NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6) . . . . .	276
NU_Getsockopt_IPV6_RECVPKTINFO (IPv6) . . . . .	278
NU_Getsockopt_IPV6_RECVRTHDR (IPv6) . . . . .	280
NU_Getsockopt_IPV6_RECVTCLASS (IPv6) . . . . .	282
NU_Getsockopt_IPV6_RTHDR (IPv6) . . . . .	284
NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6) . . . . .	286
NU_Getsockopt_IPV6_TCLASS (IPv6) . . . . .	288
NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6) . . . . .	290
NU_Getsockopt_IPV6_V6ONLY (IPv6) . . . . .	292

---

NU_Getsockopt_SO_BROADCAST (IPv4) .....	294
NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6) .....	296
NU_Getsockopt_SO_LINGER (IPv4/IPv6) .....	298
NU_Getsockopt_SO_RCVBUF (IPv4/IPv6) .....	300
NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6) .....	302
NU_Getsockopt_TCP_CFG_DSACK .....	304
NU_Getsockopt_TCP_CFG_SACK .....	306
NU_Getsockopt_TCP_CONGESTION_CTRL .....	308
NU_Getsockopt_TCP_DELAY_ACK .....	310
NU_Getsockopt_TCP_FIRST_RTO .....	312
NU_Getsockopt_TCP_FIRST_TIMEOUT .....	314
NU_Getsockopt_TCP_KEEPALIVE_R2 .....	316
NU_Getsockopt_TCP_KEEPALIVE_WAIT .....	318
NU_Getsockopt_TCP_MAX_PROBES .....	320
NU_Getsockopt_TCP_MAX_R2 .....	322
NU_Getsockopt_TCP_MAX_RTO .....	324
NU_Getsockopt_TCP_MAX_SYN_R2 .....	326
NU_Getsockopt_TCP_MSL .....	328
NU_Getsockopt_TCP_NODELAY (IPv4/IPv6) .....	330
NU_Getsockopt_TCP_PROBE_TIMEOUT .....	333
NU_Getsockopt_TCP_RCV_WINDOWSIZE .....	335
NU_Getsockopt_TCP_SND_WINDOWSIZE .....	337
NU_Getsockopt_TCP_TIMESTAMP .....	339
NU_Getsockopt_TCP_WINDOWSCALE .....	341
NU_Getsockopt_UDP_NOCHECKSUM (IPv4) .....	343
NU_IF_FreeNameIndex (IPv4/IPv6) .....	345
NU_IF_IndexToName (IPv4/IPv6) .....	346
NU_IF_NameIndex (IPv4/IPv6) .....	347
NU_IF_NameToIndex (IPv4/IPv6) .....	348
NU_Inet6_Opt_Append (IPv6) .....	349
NU_Inet6_Opt_Find (IPv6) .....	352
NU_Inet6_Opt_Finish (IPv6) .....	354
NU_Inet6_Opt_Get_Val (IPv6) .....	357
NU_Inet6_Opt_Init (IPv6) .....	358
NU_Inet6_Opt_Next (IPv6) .....	361
NU_Inet6_Opt_Set_Val (IPv6) .....	363
NU_Inet6_Rth_Add (IPv6) .....	366
NU_Inet6_Rth_GetAddr (IPv6) .....	368
NU_Inet6_Rth_Init (IPv6) .....	370
NU_Inet6_Rth_Reverse (IPv6) .....	372
NU_Inet6_Rth_Segments (IPv6) .....	374
NU_Inet6_Rth_Space (IPv6) .....	376
NU_Inet_NTOP (IPv4/IPv6) .....	378
NU_Inet_PTON (IPv4/IPv6) .....	379
NU_Init_Devices (IPv4/IPv6) .....	380
NU_Init_Net (IPv4/IPv6) .....	382
NU_Ioctl (IPv4/IPv6) .....	384
NU_Ioctl_FIONREAD (IPv4/IPv6) .....	390
NU_Ioctl_SIOCDARP (IPv4) .....	391

---

NU_Ioctl_SIOCGARP (IPv4) . . . . .	392
NU_Ioctl_SIOCGIFADDR (IPv4) . . . . .	393
NU_Ioctl_SIOCGIFADDR_IN6 (IPv6) . . . . .	395
NU_Ioctl_SIOCGIFDSTADDR (IPv4) . . . . .	397
NU_Ioctl_SIOCGIFDSTADDR_IN6 (IPv6) . . . . .	399
NU_Ioctl_SIOCGIFHWADDR . . . . .	401
NU_Ioctl_SIOCGIFNETMASK (IPv4) . . . . .	403
NU_Ioctl_SIOCGETVLAN (IPv4) . . . . .	405
NU_Ioctl_SIOCGETVLANPRIO (IPv4/IPv6) . . . . .	406
NU_Ioctl_SIOCGHWCAP (IPv4/IPv6) . . . . .	407
NU_Ioctl_SIOCICMPLIMIT . . . . .	408
NU_Ioctl_SIOCIFREQ (IPv4/IPv6) . . . . .	410
NU_Ioctl_SIOCSARP (IPv4) . . . . .	411
NU_Ioctl_SIOCSHWOPTS (IPv4/IPv6) . . . . .	413
NU_Ioctl_SIOCSIFADDR (IPv4) . . . . .	416
NU_Ioctl_SIOCSIFHWADDR . . . . .	418
NU_Ioctl_SIOCSPHYSADDR (IPv4) . . . . .	420
NU_Ioctl_SIOCSETVLAN (IPv4) . . . . .	421
NU_Ioctl_SIOCSETVLANPRIO (IPv4/IPv6) . . . . .	422
NU_IP_Multicast_Listen (IPv4) . . . . .	424
NU_IP6_Multicast_Listen (IPv6) . . . . .	427
NU_Is_Connected (IPv4/IPv6) . . . . .	430
NU_Is_IPv4_Mapped_Addr (IPv6) . . . . .	431
NU_Listen (IPv4/IPv6) . . . . .	432
NU_Ping (IPv4) . . . . .	434
NU_Ping2 (IPv4/IPv6) . . . . .	436
NU_Push (IPv4/IPv6) . . . . .	438
NU_Rarp (IPv4) . . . . .	440
NU_Recv (IPv4/IPv6) . . . . .	442
NU_Recvmsg (IPv4/IPv6) . . . . .	445
NU_Recv_From (IPv4/IPv6) . . . . .	450
NU_Recv_From_Raw (IPv4/IPv6) . . . . .	456
NU_Recv_IF_Addr (IPv4) . . . . .	460
NU_Remove_Device (IPv4/IPv6) . . . . .	462
NU_Remove_IP_From_Device (IPv4/IPv6) . . . . .	464
NU_Rip2_Initialize (IPv4) . . . . .	466
NU_Ripng_Initialize (IPv6) . . . . .	468
NU_Select (IPv4/IPv6) . . . . .	470
NU_Send (IPv4/IPv6) . . . . .	474
NU_Sendmsg (IPv4/IPv6) . . . . .	477
NU_Send_To (IPv4/IPv6) . . . . .	483
NU_Send_To_Raw (IPv4/IPv6) . . . . .	488
NU_Set_Default_Hop_Limit (IPv6) . . . . .	491
NU_Set_Default_TTL (IPv4) . . . . .	492
NU_Set_Device_Hostname . . . . .	493
NU_Set_DHCP_DUID (IPv4/IPv6) . . . . .	494
NU_Set_DHCP_IAID (IPv4/IPv6) . . . . .	496
NU_Set_Domain_Name (IPv4/IPv6) . . . . .	498
NU_Set_Host_Name (IPv4/IPv6) . . . . .	499

---

NU_Set_IP_Forwarding (IPv4/IPv6) . . . . .	500
NU_Set_Reasm_Max_Size (IPv4) . . . . .	501
NU_Shutdown (IPv4/IPv6) . . . . .	502
NU_Socket (IPv4/IPv6) . . . . .	504
NU_Setsockopt (IPv4/IPv6) . . . . .	506
NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4) . . . . .	520
NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4) . . . . .	522
NU_Setsockopt_IP_BROADCAST_IF (IPv4) . . . . .	524
NU_Setsockopt_IP_HDRINC (IPv4) . . . . .	526
NU_Setsockopt_IP_MULTICAST_IF (IPv4) . . . . .	528
NU_Setsockopt_IP_MULTICAST_TTL (IPv4) . . . . .	530
NU_Setsockopt_IP_RECVIFADDR (IPv4) . . . . .	532
NU_Setsockopt_IP_TOS (IPv4) . . . . .	534
NU_Setsockopt_IP_TTL (IPv4) . . . . .	536
NU_Setsockopt_IPV6_CHECKSUM (IPv6) . . . . .	538
NU_Setsockopt_IPV6_DSTOPTS (IPv6) . . . . .	540
NU_Setsockopt_IPV6_HOPOPTS (IPv6) . . . . .	543
NU_Setsockopt_IPV6_JOIN_GROUP (IPv6) . . . . .	546
NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6) . . . . .	548
NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6) . . . . .	550
NU_Setsockopt_IPV6_MULTICAST_IF (IPv6) . . . . .	552
NU_Setsockopt_IPV6_NEXTHOP (IPv6) . . . . .	554
NU_Setsockopt_IPV6_PKTINFO (IPv6) . . . . .	556
NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6) . . . . .	558
NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6) . . . . .	560
NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6) . . . . .	562
NU_Setsockopt_IPV6_RECVPKTINFO (IPv6) . . . . .	564
NU_Setsockopt_IPV6_RECVRTHDR (IPv6) . . . . .	566
NU_Setsockopt_IPV6_RECVTCLASS (IPv6) . . . . .	568
NU_Setsockopt_IPV6_RTHDR (IPv6) . . . . .	570
NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6) . . . . .	572
NU_Setsockopt_IPV6_TCLASS (IPv6) . . . . .	575
NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6) . . . . .	577
NU_Setsockopt_IPV6_V6ONLY (IPv6) . . . . .	579
NU_Setsockopt_SO_BROADCAST (IPv4) . . . . .	581
NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6) . . . . .	583
NU_Setsockopt_SO_LINGER (IPv4/IPv6) . . . . .	585
NU_Setsockopt_SO_RCVBUF (IPv4/IPv6) . . . . .	587
NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6) . . . . .	589
NU_Setsockopt_TCP_CFG_DSACK . . . . .	591
NU_Setsockopt_TCP_CFG_SACK . . . . .	593
NU_Setsockopt_TCP_CONGESTION_CTRL . . . . .	595
NU_Setsockopt_TCP_DELAY_ACK . . . . .	597
NU_Setsockopt_TCP_FIRST_PROBE_TIMEOUT . . . . .	599
NU_Setsockopt_TCP_FIRST_RTO . . . . .	601
NU_Setsockopt_TCP_KEEPALIVE_R2 . . . . .	603
NU_Setsockopt_TCP_KEEPALIVE_WAIT . . . . .	605
NU_Setsockopt_TCP_MAX_PROBES . . . . .	607
NU_Setsockopt_TCP_MAX_R2 . . . . .	609

NU_Setsockopt_TCP_MAX_RTO. ....	611
NU_Setsockopt_TCP_MAX_SYN_R2 .....	613
NU_Setsockopt_TCP_MSL .....	615
NU_Setsockopt_TCP_PROBE_TIMEOUT .....	617
NU_Setsockopt_TCP_NODELAY (IPv4/IPv6) .....	619
NU_Setsockopt_TCP_RCV_WINDOWSIZE .....	621
NU_Setsockopt_TCP_TIMESTAMP .....	623
NU_Setsockopt_UDP_NOCHECKSUM (IPv4) .....	625
NU_Setsockopt_TCP_WINDOWSCALE .....	627
NU_Setup_Configured_Tunnel (IPv6) .....	629
NU_Socket_Connected. ....	631
NU_Set_MDNS_Hostname_Callback .....	632
NU_Start_mDNS_Query .....	634
NU_Stop_mDNS_Query .....	636
NU_TFTPC_Get2 (IPv4/IPv6) .....	638
NU_TFTPC_Put2 (IPv4/IPv6) .....	640
NU_Test_Link_Up .....	642
NU_Update_DNS_Record .....	643
NU_Update_Route (IPv4/IPv6) .....	646
NU_ZC_Allocate_Buffer (IPv4/IPv6) .....	648
NU_ZC_BUF_LEN (IPv4/IPv6) .....	650
NU_ZC_Deallocate_Buffer (IPv4/IPv6) .....	651
NU_ZC_Recv (IPv4/IPv6) .....	653
NU_ZC_Recv_From (IPv4/IPv6) .....	657
NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6) .....	661
NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6) .....	663
NU_ZC_SEGMENT_DATA (IPv4/IPv6) .....	665
NU_ZC_SEGMENT_NEXT (IPv4/IPv6) .....	667
NU_ZC_Send (IPv4/IPv6) .....	669
NU_ZC_Send_To (IPv4/IPv6) .....	672
NET User Management .....	674
User Management API Functions .....	675
UM_Add_User .....	676
UM_Del_User .....	678
UM_Delete_User_Permissions .....	679
UM_Find_User .....	680
UM_Find_User_First .....	681
UM_Find_User_Next .....	682
UM_Get_User_Count .....	683
UM_Validate_User .....	684
NET APIs Available to a Nucleus Process .....	685
NET Shell Commands .....	691
ipconfig .....	692
Configuring Nucleus NET .....	692
Metadata Options File .....	692
Reduce the Footprint Size .....	695
Optimal Buffer Size .....	696
Multicasting .....	696
Loopback Device .....	696

---

ARP .....	696
Zero Copy Interface .....	696
Specialized Routines .....	697
Socket Options .....	697
IOCTL Options .....	697
Optimized Checksum and Block Copy Routines .....	697
Using the Optimized Block Copy Routine .....	697
Using the Optimized Assembly Checksum Routine .....	698
Using the Optimized Endian-Dependent C Checksum Routine .....	699
Inline Code .....	699
Nucleus NET Inline Routines .....	699
Ethernet Driver Inline Routines .....	700
Ethernet Drivers .....	700
Receive and Transmit Buffer Descriptors .....	700
 <b>Chapter 3</b>	
<b>NET User Manual .....</b>	<b>973</b>
NET User Overview .....	973
Operation .....	973
Initialization .....	973
Use Cases .....	974
Nucleus NET Internals .....	985
Threads .....	985
Buffer Management .....	986
Events Dispatcher .....	992
Interface Management .....	995
Error Logging .....	996
Support for Hardware Offloading .....	997
Appendix .....	999
Configuration Options .....	999
 <b>Chapter 4</b>	
<b>Network Address Translator (NAT) .....</b>	<b>1017</b>
NAT Overview .....	1017
Example NAT Router .....	1018
Multiple Private Devices .....	1018
Limited Resources .....	1019
NAT Protocol .....	1019
NAT Fragmentation and Reassembly .....	1020
NAT Files .....	1020
NAT Data Structures .....	1021
NAT_DEVICE .....	1021
NAT_PORTMAP_SERVICE .....	1022
NAT Functions .....	1022
NAT_Initialize .....	1024
NAT_Portmap .....	1026
NAT_Shutdown .....	1028
Extended Protocol Support .....	1028

Adding Support for Protocols that Contain Port Numbers . . . . .	1029
Adding Support for Protocols Without Port Numbers . . . . .	1032
Adding Additional Application Level Gateways (ALG) . . . . .	1036
. . . . .	1036

## Chapter 5

### DHCP Server . . . . . 1045

DHCP Overview . . . . .	1045
Theory of Operation . . . . .	1045
DHCP Server Requirements . . . . .	1047
Boot-up Configuration of DHCP Server. . . . .	1047
User Configurable Defines . . . . .	1049
DHCP Server User Control. . . . .	1051
Adding Supplementary Options . . . . .	1053
DHCP Data Structures . . . . .	1055
id_struct. . . . .	1055
DHCPS_CLIENT_ID . . . . .	1056
DHCP Server API Function Reference . . . . .	1056
DHCPS_Server_Reset . . . . .	1059
DHCPS_Add_Broadcast_Address . . . . .	1061
DHCPS_Add_DNS_Server . . . . .	1063
DHCPS_Add_Domain Name . . . . .	1065
DHCPS_Add_IP_Range . . . . .	1067
DHCPS_Add_Router . . . . .	1069
DHCPS_Create_Config_Entry . . . . .	1071
DHCPS_Delete_Broadcast_Address . . . . .	1073
DHCPS_Delete_Config_Entry . . . . .	1075
DHCPS_Delete_DNS_Server. . . . .	1076
DHCPS_Delete_Domain_Name. . . . .	1078
DHCPS_Delete_IP_Range . . . . .	1080
DHCPS_Delete_Router . . . . .	1082
DHCPS_Disable_Configuration. . . . .	1084
DHCPS_Disable_Ethernet_IEEE_Encapsulation. . . . .	1086
DHCPS_Disable_IP_Forwarding . . . . .	1087
DHCPS_Disable_TCP_Keepalive_Garbage . . . . .	1088
DHCPS_Disable_Trailer_Encapsulation . . . . .	1089
DHCPS_Enable_Configuration . . . . .	1090
DHCPS_Enable_Ethernet_IEEE_Encapsulation . . . . .	1091
DHCPS_Enable_IP_Forwarding . . . . .	1092
DHCPS_Enable_TCP_Keepalive_Garbage . . . . .	1093
DHCPS_Enable_Trailer_Encapsulation. . . . .	1094
DHCPS_Get_ARP_Cache_Timeout. . . . .	1095
DHCPS_Get_Broadcast_Address. . . . .	1097
DHCPS_Get_Config_Entry . . . . .	1099
DHCPS_Get_Default_Lease_Time . . . . .	1100
DHCPS_Get_DNS_Servers . . . . .	1102
DHCPS_Get_Domain_Name . . . . .	1104
DHCPS_Get_IP_Range . . . . .	1106



---

DHCPS_Get_IP_TTL .....	1108
DHCPS_Get_Offered_Wait_Time .....	1110
DHCPS_Get_Rebind_Time .....	1112
DHCPS_Get_Renewal_Time .....	1114
DHCPS_Get_Router .....	1116
DHCPS_Get_Subnet_Address .....	1118
DHCPS_Get_Subnet_Mask .....	1120
DHCPS_Get_TCP_Keepalive_Interval .....	1122
DHCPS_Get_TCP_TTL .....	1124
DHCPS_Set_ARP_Cache_Timeout .....	1126
DHCPS_Set_Default_Lease_Time .....	1127
DHCPS_Set_IP_TTL .....	1129
DHCPS_Set_Offered_Wait_Time .....	1130
DHCPS_Set_Rebind_Time .....	1132
DHCPS_Set_Renewal_Time .....	1134
DHCPS_Set_Subnet_Address .....	1136
DHCPS_Set_Subnet_Mask .....	1138
DHCPS_Set_TCP_Keepalive_Interval .....	1140
DHCPS_Set_TCP_TTL .....	1141
DHCPS_Shutdown_Server .....	1142

## Chapter 6

<b>Security Sockets Layer (SSL) .....</b>	<b>1171</b>
SSL Overview .....	1171
SSL Functionality .....	1173
Creating an Application .....	1173
SSL Initialization .....	1173
Connecting to a Server .....	1173
Accepting Connections .....	1174
Transferring Data .....	1175
Terminating the Session .....	1176
Real-Time Clock .....	1177
Authentication vs. Encryption .....	1177
SSL Authentication .....	1177
SSL Encryption .....	1178
SSL Example .....	1178
SSL Packet .....	1179
Session Negotiation .....	1179
Data Transfer .....	1182
Message Authentication .....	1182
Encryption .....	1182
SSL Configuration File .....	1182
nu_openssl_cfg.h .....	1182
SSL Lite (CyaSSL) .....	1183

## Chapter 7

<b>IPsec .....</b>	<b>1189</b>
IPsec Introduction .....	1189



---

Nucleus IPsec . . . . .	1190
Encapsulating Security Payload (ESP) . . . . .	1191
ESP Packet . . . . .	1191
ESP Header . . . . .	1193
ESP Trailer . . . . .	1193
Authentication Header . . . . .	1193
Supported Authentication Algorithms . . . . .	1193
AH Packet . . . . .	1194
AH Header . . . . .	1194
Encryptor . . . . .	1195
Encryption Algorithms Supported . . . . .	1195
IPsec Configurations . . . . .	1195
Security Databases . . . . .	1196
Security Association . . . . .	1196
Security Policy and Security Policy Database . . . . .	1196
Security Policy Group . . . . .	1197
Security Policy . . . . .	1197
Policy Sorting Order . . . . .	1197
IPsec Data Structures . . . . .	1198
IPSEC_POLICY . . . . .	1198
IPSEC_SECURITY_PROTOCOL . . . . .	1201
IPSEC_SELECTOR . . . . .	1202
IPSEC_IP_ADDR . . . . .	1205
IPSEC_BUNDLE_LIST . . . . .	1205
IPSEC_SA_LIFETIME . . . . .	1206
IPSEC_OUTBOUND_SA . . . . .	1207
IPSEC_OUTBOUND_INDEX . . . . .	1208
IPSEC_INBOUND_SA . . . . .	1209
IPSEC_INBOUND_INDEX . . . . .	1210
IPsec API Functions . . . . .	1212
IPSEC_Add_Group . . . . .	1213
IPSEC_Add_Inbound_SA	
IPSEC_Add_Inbound_ESN_SA . . . . .	1214
IPSEC_Add_Outbound_SA	
IPSEC_Add_Outbound_ESN_SA . . . . .	1217
IPSEC_Add_Policy . . . . .	1220
IPSEC_Add_To_Group . . . . .	1223
IPSEC_Apply_To_Interface . . . . .	1225
IPSEC_Get_Group . . . . .	1227
IPSEC_Get_Group_Opt . . . . .	1229
IPSEC_Get_Inbound_SA_Opt . . . . .	1232
IPSEC_Get_Outbound_SA_Opt . . . . .	1240
IPSEC_Get_Policy_Index . . . . .	1247
IPSEC_Get_Policy_Opt . . . . .	1249
IPSEC_Remove_From_Group . . . . .	1255
IPSEC_Remove_Group . . . . .	1256
IPSEC_Remove_Inbound_SA . . . . .	1257
IPSEC_Remove_Outbound_SA . . . . .	1259
IPSEC_Remove_Policy . . . . .	1261

IPSEC_Set_Group_Opt .....	1263
IPSEC_Set_Policy_Opt .....	1265
Customizations.....	1269
Support for New Authentication Algorithms .....	1269
Support for New Encryption Algorithms .....	1270
Nucleus IPsec Processing .....	1271
Outbound Processing .....	1271
Inbound Processing.....	1272

## Chapter 8

### IPsec-IKE..... 1323

IKEv1 Exchange Overview .....	1323
IKEv2 Exchange Overview .....	1325
Nucleus IPsec - IKE Protocol .....	1326
Supported RFCs .....	1326
Supported Algorithms and Protocols .....	1326
Exchange Modes for IKEv1 .....	1327
IKEv2 Exchanges .....	1327
Configurable Macros for IKEv1 .....	1329
IKE_TIMEOUT .....	1329
IKE_ENABLE_BLOCKING_REQUEST .....	1329
IKE_MAX_WAIT_EVENTS.....	1329
IKE_DEBUG .....	1329
IKE_DEBUG_LOG .....	1330
IKE_INCLUDE_VERSION_2.....	1330
IKE_INCLUDE_MAIN_MODE .....	1330
IKE_INCLUDE_AGGR_MODE.....	1330
IKE_INCLUDE_INFO_MODE.....	1330
IKE_INCLUDE_PSK_AUTH .....	1330
IKE_INCLUDE_SIG_AUTH.....	1330
IKE_ENABLE_INITIAL_CONTACT.....	1331
IKE_ENABLE_DOMAIN_NAME_ID .....	1331
IKE_RETAIN_LAST_MESSAGE.....	1331
IKE_PHASE1_DEFAULT_XCHG .....	1331
IKE_INCLUDE_MODP_1024.....	1331
IKE_INCLUDE_MODP_1536.....	1332
IKE_INCLUDE_MODP_2048.....	1332
IKE_INCLUDE_MODP_3072.....	1332
IKE_INCLUDE_MODP_4096.....	1332
IKE_INCLUDE_MODP_6144.....	1332
IKE_INCLUDE_MODP_8192.....	1332
IKE_INCLUDE_MODP_1024_160_POS .....	1332
IKE_INCLUDE_MODP_2048_224_POS .....	1333
IKE_INCLUDE_MODP_2048_256_POS .....	1333
IKE_SOFT_LIFETIME_OFFSET .....	1333
IKE_RESEND_INTERVAL .....	1333
IKE_RESEND_COUNT.....	1333
IKE_PHASE1_TIMEOUT.....	1333

---

IKE_PHASE2_TIMEOUT .....	1333
IKE_MAX_DELETE_SPI .....	1334
IKE_MAX_PROPOSALS .....	1334
IKE_MAX_TRANSFORMS .....	1334
IKE_MAX_DOMAIN_NAME_LEN .....	1334
IKE_MAX_GROUP_NAME_LEN .....	1334
IKE_OUTBOUND_NONCE_DATA_LEN .....	1334
IKE_MAX_INBOUND_PACKET_LEN .....	1335
IKE_MAX_OUTBOUND_PACKET_LEN .....	1335
IKE_MAX_BUFFERS .....	1335
IKE_TASK_STACK_SIZE .....	1335
IKE_TASK_PRIORITY .....	1335
IKE_TASK_TIME_SLICE .....	1335
IKE_TASK_PREEMPT .....	1335
IKE_EVENT_TASK_STACK_SIZE .....	1336
IKE_EVENT_TASK_PRIORITY .....	1336
IKE_EVENT_TASK_TIME_SLICE .....	1336
IKE_EVENT_TASK_PREEMPT .....	1336
IKE_INCLUDE_DES .....	1336
IKE_INCLUDE_3DES .....	1336
IKE_INCLUDE_BLOWFISH .....	1336
IKE_INCLUDE_CAST128 .....	1337
IKE_INCLUDE_AES .....	1337
IKE_INCLUDE_MD5 .....	1337
IKE_INCLUDE_SHA1 .....	1337
IKE_INCLUDE_RSA .....	1337
IKE_INCLUDE_PEM .....	1337
IKE_INCLUDE_CRL_SUPPORT .....	1338
IKE_MAX_ENCRYPT_BLOCK_LEN .....	1338
IKE_MAX_HASH_DATA_LEN .....	1338
Compile Time Options for IKEv2 .....	1338
IKE2_SA_TIMEOUT .....	1338
IKE2_ENABLE_NAT_TRAVERSAL .....	1338
IKE2_MAX_PACKET_LEN .....	1338
IKE2_MESSAGE_REPLY_TIMEOUT .....	1339
IKE Database .....	1339
IKE Policy .....	1340
IKE Data Structures .....	1341
IKE_POLICY .....	1342
IKE_POLICY_SELECTOR/IKE_IDENTIFIER .....	1348
IKE_SELECTOR_IP .....	1350
IKE_IPS_ID .....	1351
IKE_ATTRIB .....	1351
IKE_SA_LIFETIME .....	1355
IKE_KEY_PAIR .....	1356
IKE_PRESHARED_KEY .....	1357
IKE_INITIATE_REQ .....	1357
Pre-shared Keys .....	1358
IKE API Functions .....	1360

---

IKE_Add_Group.....	1362
IKE_Add_Policy.....	1364
IKE_Add_Preshared_Key.....	1370
IKE_Add_To_Group.....	1373
IKE_Get_Group.....	1375
IKE_Get_Group_Opt.....	1377
IKE_Get_Policy_Index.....	1380
IKE_Get_Policy_Opt.....	1382
IKE_Get_Preshared_Key_Index.....	1385
IKE_Initiate.....	1387
IKE_Remove_From_Group.....	1393
IKE_Remove_Group.....	1394
IKE_Remove_Policy.....	1396
IKE_Remove_Preshared_Key.....	1398
IKE_Set_Policy_Opt.....	1400
IKE_Shutdown.....	1403
IKE API Usage Example.....	1404
IKE Environment.....	1404
IKE Code Samples.....	1406
IKE Customizations.....	1414
Adding Phase 1 Encryption Algorithm.....	1414
Adding Phase 1 Hash Algorithm.....	1417
Adding Phase 2 Encryption Algorithm.....	1420
Adding Phase 2 Hash Algorithm.....	1422
IKE Proposals.....	1426
Exchange Invocation.....	1427
IPsec SA Re-keying.....	1428
SA Re-keying with Microsoft Windows.....	1429
IKE Error Codes.....	1429

## Chapter 9

### Simple Network Management Protocol (SNMP)..... 1493

SNMP Overview.....	1493
Dispatcher.....	1494
Message Processing Subsystem.....	1494
Security Subsystem.....	1494
Access Control Subsystem.....	1495
Applications Subsystem.....	1495
Management Information Base (MIB) Engines.....	1495
Configuring SNMP Defines.....	1496
include/networking/snmp_prt.h.....	1496
include/networking/snmp.h.....	1498
snmp_cfg.h.....	1498
Configuring NET for SNMP Support.....	1501
Configuring Nucleus SNMP.....	1501
SNMPv1.....	1502
SNMPv3.....	1502
SNMP Configuration through the Serial Interface.....	1503

---

SNMP Data Structures . . . . .	1505
mib_element_t . . . . .	1506
mib_callback_t . . . . .	1507
CBSM_COMMUNITY_STRUCT . . . . .	1507
SNMP_CBSM_COMMUNITY_STRUCT . . . . .	1509
SNMP_NOTIFY_REQ_STRUCT . . . . .	1510
SNMP_NOTIFY_TABLE . . . . .	1511
SNMP_NOTIFY_FILTER_PROFILE_TABLE . . . . .	1512
SNMP_NOTIFY_FILTER_TABLE . . . . .	1514
SNMP_SS_STRUCT . . . . .	1516
SNMP_TARGET_ADDRESS_TABLE . . . . .	1517
SNMP_TARGET_PARAMS_TABLE . . . . .	1519
SNMP_USM_USER_STRUCT . . . . .	1521
USM_USERS_STRUCT . . . . .	1522
USM_PRIV_PROT_STRUCT . . . . .	1524
USM_AUTH_PROT_STRUCT . . . . .	1525
VACM_CONTEXT_STRUCT . . . . .	1526
VACM_SEC2GROUP . . . . .	1526
VACM_SEC2GROUP_STRUCT . . . . .	1527
VACM_ACCESS . . . . .	1529
VACM_ACCESS_STRUCT . . . . .	1530
VACM_VIEWTREE . . . . .	1532
VACM_VIEWTREE_STRUCT . . . . .	1534
SNMP API Functions . . . . .	1536
NU_SNMP_Initialize . . . . .	1539
SNMP_Configuration . . . . .	1540
NU_SNMP_Set_Host_Id . . . . .	1541
NU_SNMP_Set_Host_Id6 . . . . .	1543
SNMP_Mib_Register . . . . .	1544
SNMP_Mib_Unregister . . . . .	1546
SNMP_Get_Notification_Ptr . . . . .	1547
SNMP_Notification_Ready . . . . .	1549
AgentGetAuthenTraps . . . . .	1551
AgentSetAuthenTraps . . . . .	1552
AgentSetColdTraps . . . . .	1553
SNMP_Get_Engine_ID . . . . .	1554
CBSM_Add_Community . . . . .	1555
CBSM_Find_Community_Index . . . . .	1557
CBSM_Remove_Community . . . . .	1560
CBSM_Save_Community . . . . .	1562
SNMP_Add_To_Notify_Tbl . . . . .	1564
SNMP_Find_Notify_Entry . . . . .	1566
SNMP_Remove_From_Notify_Table . . . . .	1568
SNMP_Add_To_Profile_Tbl . . . . .	1570
SNMP_Find_Profile_Entry . . . . .	1572
SNMP_Remove_From_Profile_Table . . . . .	1574
SNMP_Add_To_Filter_Tbl . . . . .	1576
SNMP_Find_Filter_Entry . . . . .	1578
SNMP_Remove_From_Filter_Table . . . . .	1580

---

SNMP_Add_Target .....	1582
SNMP_Find_Target .....	1584
SNMP_Remove_From_Tgr_Table.....	1586
SNMP_Add_Params.....	1588
SNMP_Find_Params .....	1590
SNMP_Remove_From_Params_Table.....	1592
USM_Add_User .....	1594
USM_Lookup_Users .....	1597
USM_Remove_User.....	1600
USM_Save_User.....	1603
USM_Lookup_Priv_Prot .....	1606
USM_Lookup_Auth_Prot.....	1609
VACM_Add_Context.....	1612
VACM_Search_Context.....	1614
VACM_Remove_Context.....	1616
VACM_InsertGroup.....	1617
VACM_Search_Group .....	1619
VACM_Remove_Group.....	1621
VACM_Save_Group.....	1623
VACM_InsertAccessEntry .....	1625
VACM_Search_AccessEntry .....	1627
VACM_Remove_AccessEntry.....	1630
VACM_Save_Access .....	1632
VACM_InsertMibView .....	1634
VACM_Search_MibView .....	1637
VACM_Remove_MibView .....	1640
VACM_Save_View .....	1643
Nucleus MIB-II Defines .....	1645
RFC1213 Defines File .....	1645
MIB-II: Managed Objects .....	1646
The System Group .....	1646
The Interfaces Group .....	1647
The Interface Extension Group.....	1649
The Interface Stack Group .....	1655
The Interface Receive Address Group .....	1655
The IP Group.....	1656
The ipRouteTable .....	1658
The ipNetToMediaTable.....	1660
The ICMP Group .....	1661
The TCP Group.....	1662
The UDP Group .....	1664
The EGP Group.....	1665
The SNMP Group .....	1666
Nucleus SNMPv3 MIB .....	1669
Nucleus SNMPv3 MIB: Defines .....	1669
snmp_cfg.h .....	1669
Nucleus SNMPv3 MIB Managed Objects.....	1670
SNMP Engine Group .....	1670
SNMP MPD Stats Group .....	1671

---

SNMP Target Address Table .....	1672
SNMP Target Params Table .....	1673
SNMP Notify Table .....	1674
SNMP Notify Filter Profile Table .....	1674
SNMP Notify Filter Table .....	1675
USM User Group .....	1675
VACM Context Group .....	1677
VACM Security To Group .....	1678
VACM Access Group .....	1678
VACM View Tree Family Group .....	1680
SNMP Community Table .....	1681
Nucleus MIB Utility .....	1682
Generating Source Code Using the MIB Utility .....	1683
Incorporating Generated MIB Source Code into SNMP Agent .....	1684
PPP MIB .....	1686
PPP MIB: Managed Objects .....	1686
Advanced SNMP Topics .....	1693
Security Subsystem Configuration .....	1693
Adding a New SNMP Security Model .....	1693
SNMP Configurations .....	1696
include/networking/snmp_cfg.h .....	1696
networking/snmp/snmp_cfg.c .....	1697
User-based Security Model .....	1697
Authentication .....	1697
Timeliness .....	1697
Privacy .....	1697
Message Format .....	1697
Discovery .....	1698
Key Management .....	1698
include/networking/snmp_cfg.h .....	1698
networking/snmp/snmp_cfg.c .....	1700
Password to Key Utility .....	1701
Community-Based Security Model .....	1703
include/networking/snmp_cfg.h .....	1704
networking/snmp/snmp_cfg.c .....	1704
Application Subsystem .....	1705
Protocol Operations .....	1705
Access Control Subsystem Configuration .....	1708
Elements of VACM Model .....	1708
Inputs to Access Control Subsystem .....	1710
Outputs of Access Control Subsystem .....	1711
Access Control Subsystem Processing .....	1711
include/networking/snmp_cfg.h .....	1713
networking/snmp/snmp_cfg.c .....	1714
Notification Originator Application .....	1715
Notify Table .....	1715
Target Address Table .....	1716
Target Params Table .....	1716
Notify Filter Profile Table .....	1717

Notify Filter Table .....	1717
Optimizing Buffer Sizes .....	1718
Get Bulk .....	1718
Get Next .....	1718
SNMPv1 Security Configurations .....	1719
VACM MIB View Configuration .....	1721
How to Use VACM Views - VACM Access Table .....	1726
Access Entries - The Security to Group Table .....	1730
Mapping Community Names into the Community Table .....	1731
Adding Security Using the Target Address Table .....	1740
SNMPv2c Security Configurations .....	1753
VACM MIB View .....	1755
How to Use VACM Views - VACM Access Table .....	1755
How to Use Access Entries - Security to Group Table .....	1759
Mapping Everything with Community Name in the Community Table .....	1760
Additional Security by CBSM - Target Address Table .....	1770
SNMPv3 Security Configurations .....	1784
Enhancements and API Changes .....	1790
Nucleus SNMP 2.1 .....	1790
Nucleus SNMP 2.2 .....	1792
Nucleus SNMP 2.3 .....	1795
Nucleus SNMP 2.4 .....	1796

## Chapter 10

### Extended Protocol (XPROT).....1987

XPROT Overview .....	1987
FTP Server .....	1987
FTP Client .....	1987
Telnet Server .....	1987
Telnet Client .....	1988
Trivial File Transfer Protocol Server .....	1988
FTP Package .....	1988
FTP Commands .....	1988
FTP Common Defines and Data Structures .....	1991
FTP Defines .....	1991
FTP Data Structures .....	1993
FTP Server-Level Defines and Data Structures .....	1994
FTP Server-Level Defines .....	1994
FTP Server-Level Data Structures .....	1995
FTP Server Simple User Authentication .....	1995
Simple User Authentication Defines .....	1995
Simple User Authentication Data Structures .....	1996
Simple User Authentication Configurable Variable .....	1997
FTP Server-Level Functions .....	1997
NU_FTP_Server_Init .....	1998
NU_FTP_Server_Uninit .....	1999
FTP Client-Level Defines and Data Structures .....	2001
FTP Client-Level Defines .....	2001



---

FTP Client-Level Data Structures . . . . .	2001
FTP Client-Level Functions . . . . .	2003
NU_FTPC_Client_Append_To_File . . . . .	2005
NU_FTPC_Client_ChDir . . . . .	2007
NU_FTPC_Client_Close . . . . .	2009
NU_FTPC_Client_Dir . . . . .	2011
NU_FTPC_Client_Get . . . . .	2013
NU_FTPC_Client_Login . . . . .	2015
NU_FTPC_Client_MkDir . . . . .	2017
NU_FTPC_Client_Nlist . . . . .	2019
NU_FTPC_Client_Open . . . . .	2021
NU_FTPC_Client_Open2 . . . . .	2023
NU_FTPC_Client_Put . . . . .	2025
NU_FTPC_Client_Rename_File . . . . .	2028
NU_FTPC_Client_Restart . . . . .	2030
NU_FTPC_Client_Rmdir . . . . .	2032
NU_FTPC_Client_Size . . . . .	2034
NU_FTPC_Client_Status . . . . .	2036
NU_FTPC_Client_Tran_Mode . . . . .	2038
NU_FTPC_Client_Tran_Type . . . . .	2040
FTP Client-Level Command Primitives . . . . .	2041
NU_FCP_Client_ACCT . . . . .	2043
NU_FCP_Client_APPE . . . . .	2045
NU_FCP_Client_CDUP . . . . .	
NU_FCP_Client_XCUP . . . . .	2047
NU_FCP_Client_CWD . . . . .	
NU_FCP_Client_XCWD . . . . .	2049
NU_FCP_Client_DELE . . . . .	2051
NU_FCP_Client_EPRT . . . . .	2053
NU_FCP_Client_EPSV . . . . .	2055
NU_FCP_Client_FEAT . . . . .	2057
NU_FCP_Client_HELP . . . . .	2059
NU_FCP_Client_LIST . . . . .	2061
NU_FCP_Client_MKD . . . . .	
NU_FCP_Client_XMKD . . . . .	2063
NU_FCP_Client_MODE . . . . .	2065
NU_FCP_Client_NLST . . . . .	2067
NU_FCP_Client_NOOP . . . . .	2069
NU_FCP_Client_PASS . . . . .	2071
NU_FCP_Client_PASV . . . . .	2073
NU_FCP_Client_PORT . . . . .	2075
NU_FCP_Client_PWD . . . . .	
NU_FCP_Client_XPWD . . . . .	2077
NU_FCP_Client_QUIT . . . . .	2079
NU_FCP_Client_REST . . . . .	2081
NU_FCP_Client_RETR . . . . .	2083
NU_FCP_Client_RMD . . . . .	
NU_FCP_Client_XRMD . . . . .	2085
NU_FCP_Client_RNFR . . . . .	2087

---

NU_FCP_Client_RNTO .....	2089
NU_FCP_Client_STAT .....	2091
NU_FCP_Client_STOR .....	2093
NU_FCP_Client_STRU .....	2095
NU_FCP_Client_SYST .....	2097
NU_FCP_Client_Tran_Ack .....	2099
NU_FCP_Client_TYPE .....	2101
NU_FCP_Client_USER .....	2103
NU_FCP_Client_Verify_Caller .....	2105
NU_FCP_Reply_Read .....	2107
Telnet .....	2109
Telnet Defines .....	2109
Telnet Negotiation Tables .....	2109
Telnet Data Structures .....	2110
NU_TN_PARAMETERS .....	2110
Telnet Service Call Primitives .....	2111
NU_Close_And_Check_Retval .....	2113
NU_Install_Negotiate_Options .....	2114
NU_Received_Exit .....	2115
NU_Telnet_Check_Connection .....	2116
NU_Telnet_Client_Connect .....	2118
NU_Telnet_Client_Connect2 .....	2119
NU_Telnet_Echo_Get .....	2121
NU_Telnet_Echo_Set .....	2123
NU_Telnet_Free_Parameters .....	2125
NU_Telnet_Init_Parameters .....	2126
NU_Telnet_Send .....	2127
NU_Telnet_Server_Accept .....	2128
NU_Telnet_Socket .....	2129
NU_Telnet_Specific_Negotiate .....	2131
NU_Telnet_Start_Negotiate .....	2133
Telnet Internal Service Calls .....	2135
NU_Receive_NVT_Key .....	2136
NU_Send_NVT_Key .....	2138
NU_Telnet_Client_Init_Parameters .....	2139
NU_Telnet_Do_Negotiation .....	2141
NU_Telnet_Get_Filtered_Char .....	2143
NU_Telnet_Get_Session_Parameters .....	2145
NU_Telnet_Parse .....	2147
NU_Telnet_Pick_Up_Ascii .....	2149
NU_Telnet_Server_Init_Parameters .....	2151
NU_Wait_For_Pattern .....	2153
TFTP Server .....	2156
TFTP Server Defines and Data Structures .....	2156
TFTP Server Defines .....	2156
TFTP Server Data Structures .....	2158
TFTP Accepting and Processing Options .....	2158
Turning Off Options Processing .....	2158
TFTP Server Functions .....	2158

TFTP_Server_Init .....	2159
TFTP_Server_Uninit. ....	2160
TFTP Server Shell Commands.....	2161
tftp .....	2162
NET Notification Module .....	2163
NET Notification Module Data Structures .....	2163
Routine for sending Informational Messages to the Application .....	2164
Debugging Module for Receiving Messages from the Application .....	2165
 <b>Chapter 11</b>	
<b>HTTP Lite .....</b>	<b>2265</b>
Nucleus ReadyStart HTTP Lite Overview .....	2265
Protocol Overview .....	2265
Coverage and Limitations.....	2265
Directory Structure.....	2266
User Configuration Options.....	2266
HTTP Lite Functionality .....	2267
Plug-ins Parsing Data and Responding to Clients .....	2268
Plug-in Usage .....	2269
Example of Parsing Data From a POST Command .....	2269
HTTP Lite Data Structures .....	2271
NU_HTTP_CLIENT.....	2271
NU_HTTP_SSL_STRUCT .....	2272
HTTP_SVR_SESSION_STRUCT.....	2273
HTTP_TOKEN_INFO .....	2274
HTTP Lite Function APIs .....	2275
NU_HTTP_Lite_Client_Get.....	2277
NU_HTTP_Lite_Client_Post .....	2279
NU_HTTP_Lite_Client_Put .....	2281
NU_HTTP_Lite_Client_Delete .....	2283
NU_HTTP_Lite_Client_Init.....	2285
NU_HTTP_Lite_Server_Init .....	2286
NU_HTTP_Lite_Configure_SSL.....	2287
NU_HTTP_Lite_Server_Shutdown .....	2290
NU_HTTP_Lite_Create_Session_Handle .....	2291
NU_HTTP_Lite_Delete_Session_Handle .....	2292
NU_HTTP_Lite_Register_Plugin.....	2293
NU_HTTP_Lite_Remove_Plugin.....	2295
NU_HTTP_Lite_Create_File .....	2296
NU_HTTP_Lite_Write_File.....	2297
NU_HTTP_Lite_Delete_File .....	2298
NU_HTTP_Lite_Register_Upgrade_Plugin.....	2299
NU_HTTP_Lite_Remove_Upgrade_Plugin.....	2300
 <b>Chapter 12</b>	
<b>WebServ.....</b>	<b>2301</b>
Web Server Overview .....	2301
Hyper Text Transfer Protocol.....	2301

---

Hyper Text Mark-up Language .....	2302
WebServ Data Structures .....	2303
WS_REQUEST .....	2303
WebServ Functions .....	2304
CFS_Compress .....	2306
CFS-Decompress .....	2308
HTTP_Cookie_Value_By_Name .....	2310
HTTP_Make_Mime_Header .....	2311
HTTP_Redirect_Client .....	2312
HTTP_Send_Status_Message .....	2313
HTTP_Set_Cookie .....	2315
HTTP_Token_Value_by_Name .....	2316
HTTP_Token_Value_by_Number .....	2317
HTTP_Uri_To_Url .....	2318
HTTPS_URI_To_URL .....	2319
WS_Find_Token_by_Case .....	2320
WS_In_String .....	2321
WS_Register_Plugin .....	2322
WS_Webserv_Initialize .....	2323
WS_Set_Auth_Callback .....	2324
WSC_Get_Use_Hostname .....	2325
WSC_Set_Use_Hostname .....	2326
WSF_File_Request_Status .....	2327
WSF_Read_File .....	2329
WSF_Write_File_System .....	2330
WSN_Read_Net .....	2331
WSN_Write_To_Net .....	2333
 <b>Chapter 13</b>	
<b>WebSocket .....</b>	<b>2355</b>
WebSocket Implementation Overview .....	2355
Limitations of this WebSocket Implementation .....	2355
User Configuration Options .....	2355
Directory Structure .....	2356
Creating a WebSocket Handle from the Application .....	2356
WebSocket/HTTP Dual Stack Implementation .....	2356
HTTP Glue Layer .....	2357
WebSocket Application Interface for Sending/Receiving Messages .....	2357
Sending Messages .....	2357
Receiving Messages .....	2358
Closing the Connection .....	2358
Sending PING Frames .....	2358
WebSocket Socket Options .....	2359
WebSocket Data Structures .....	2359
NU_WSOX_CONTEXT_STRUCT .....	2359
WebSocket APIs .....	2362
NU_WSOX_Create_Context .....	2363
NU_WSOX_Accept .....	2365

NU_WSOX_Send . . . . .	2367
NU_WSOX_Recv . . . . .	2369
NU_WSOX_Toggle_Recv . . . . .	2371
NU_WSOX_Close . . . . .	2373
NU_WSOX_Schedule_Ping . . . . .	2375
NU_WSOX_Setsockopt . . . . .	2377
NU_WSOX_Getsockopt . . . . .	2379
NU_WSOX_Process_Upgrade_Request . . . . .	2381

## Chapter 14

### Simple Network Time Protocol (SNTP) . . . . . 2385

SNTP Overview . . . . .	2385
NTP/SNTP Servers . . . . .	2385
SNTP Interaction with Nucleus MMU . . . . .	2386
SNTP Client Data Structures . . . . .	2386
SNTPC_TIME . . . . .	2386
SNTPC_SERVER . . . . .	2387
Nucleus SNTP Client - API Interface . . . . .	2388
SNTPC_Add_Server . . . . .	2389
SNTPC_Delete_Server . . . . .	2391
SNTPC_Purge_Server_List . . . . .	2392
SNTPC_Server_Query . . . . .	2393
SNTPC_Set_Timezone . . . . .	2394
SNTPC_Get_Timezone . . . . .	2395
SNTPC_Get_Time . . . . .	2396
SNTPC_Get_Time_From_Server . . . . .	2397

## Chapter 15

### Simple Mail Transfer Protocol (SMTP) . . . . . 2405

SMTP Overview . . . . .	2405
SMTP Protocol . . . . .	2405
SMTP RFCs Overview . . . . .	2405
SMTP Protocol Extensions . . . . .	2405
Nucleus SMTP Client Overview . . . . .	2406
Nucleus SMPT Authentication . . . . .	2407
Nucleus SMTP Data Encryption . . . . .	2407
Attachments and Multi-Part Messages . . . . .	2407
Nucleus SMTP Supported RFCs . . . . .	2407
Configuration Macros . . . . .	2407
SMTP Client Theory of Operation . . . . .	2408
SMTP Session . . . . .	2408
SMTP Message . . . . .	2409
High-Level API Control Flow . . . . .	2409
Session Connect Sequence . . . . .	2411
Send Message Sequence . . . . .	2413
SMTP Sizes and Limits . . . . .	2413
SMTP Client High-Level API Functions . . . . .	2414
NU_SMTP_Msg_Add_Attachment . . . . .	2415

---

NU SMTP_Msg_Add_Body .....	2417
NU SMTP_Msg_Add_Recipient .....	2418
NU SMTP_Msg_Delete .....	2420
NU SMTP_Msg_Init .....	2421
NU SMTP_Msg_Set_Subject .....	2422
NU SMTP_Msg_Set_Sender .....	2423
NU SMTP_Send_Message .....	2424
NU SMTP_Session_Connect .....	2426
NU SMTP_Session_Disconnect .....	2427
NU SMTP_Session_Delete .....	2428
NU SMTP_Session_Init .....	2429
NU SMTP_Session_Set_Login .....	2431
SMTP Client Low-Level API Functions .....	2431
NU SMTP_Auth .....	2433
NU SMTP_Base_Cmd .....	2434
NU SMTP_Close .....	2436
NU SMTP_Data .....	2437
NU SMTP_Ehlo .....	2438
NU SMTP_Helo .....	2439
NU SMTP_Mail .....	2440
NU SMTP_Msg .....	2441
NU SMTP_Quit .....	2443
NU SMTP_Rcpt .....	2444
NU SMTP_Rset .....	2445
NU SMTP_StartTLS .....	2446
SMTP_Client_Init .....	2447
SMTP_Base64_Encode .....	2448
SMTP_Set_Extension_Settings .....	2449

## Chapter 16

<b>JSON Generator .....</b>	<b>2479</b>
JSON Generator Overview .....	2479
Nucleus JSON Generator Implementation .....	2479
JSON Generator Data Structures and Constants .....	2479
Interface in Action .....	2480
Sample JSON Document .....	2480
Sample JSON Generator Code .....	2481
JSON Generator APIs .....	2483
NU_JSON_Generator_Create .....	2484
NU_JSON_Generator_Destroy .....	2485
NU_JSON_Generator_Start-Token .....	2486
NU_JSON_Generator_End-Token .....	2488
NU_JSON_Generator_Add_Name .....	2490
NU_JSON_Generator_Add_String .....	2492
NU_JSON_Generator_Add_Boolean .....	2495
NU_JSON_Generator_Add_Int .....	2497
NU_JSON_Generator_Add_UInt .....	2499
NU_JSON_Generator_Add_Float .....	2501

NU_JSON_Generator_Add_Null .....	2503
NU_JSON_Generator_Get_Buffer .....	2505
NU_JSON_Generator_Clear_Buffer .....	2507

## Chapter 17

### JSON Parser ..... 2509

JSON Parser Overview .....	2509
Nucleus JSON Parser Implementation .....	2509
Understanding the Stack Levels .....	2511
JSON Parser Data Structures and Constants .....	2512
JSON_PARSER_HANDLE .....	2513
JSON_STRING .....	2513
JSON Parser APIs .....	2514
NU_JSON_Parser_Create .....	2515
NU_JSON_Parser_Destroy .....	2516
NU_JSON_Parser_Set_Data .....	2517
NU_JSON_Parser_Reset .....	2519
NU_JSON_Parser_Next .....	2520
NU_JSON_Parser_Get_Info .....	2522
NU_JSON_Get_Boolean .....	2524
NU_JSON_Get_Int .....	2526
NU_JSON_Get_UInt .....	2528
NU_JSON_Get_Null .....	2530
NU_JSON_Get_String .....	2532
NU_JSON_Get_Float .....	2534

## Chapter 18

### Wireless Protected Access (WPA) ..... 2537

WLAN Overview .....	2537
Nucleus WPA Supplicant .....	2537
Key Features and Supported Standards .....	2538
WPA Supplicant Architecture .....	2538
Open Source “wpa_supplicant” Project .....	2539
Nucleus WPA Supplicant API Functions .....	2540
WPA_Supplicant_Reload_Config .....	2541
WPA_Supplicant_Get_Iface_Count .....	2542
Nucleus WPA Supplicant Configuration .....	2542
.metadata .....	2543
wpa_sup.cfg .....	2544
wpa_supplicant_cfg.c .....	2544
wpa_supplicant_cfg.h .....	2545
EAP-TLS Authentication Setup .....	2546
Configuring the Nucleus Side for EAP-TLS .....	2546
Configuration of a RADIUS Server and Ethernet Switch for EAP-TLS .....	2547
Setting Up a RADIUS Server .....	2548
Configuring FreeRADIUS Server for Windows XP .....	2548
Configuring the WPA Supplicant-Capable Ethernet Switch .....	2550

---

## Chapter 19

### Point-To-Point Protocol (PPP)..... 2557

PPP Overview .....	2557
Required PPP Include File .....	2557
Nucleus PPP Driver Modules .....	2558
Applications .....	2558
Nucleus PPP Module .....	2560
PPP Data Structures .....	2560
NU_PPP_OPTIONS .....	2560
NU_PPP_CFG .....	2562
PPP Services .....	2564
NU_Dial_PPP_Server .....	2566
NU_PPP_Hangup .....	2568
NU_Set_PPP_Login .....	2570
NU_Add_PPP_User .....	2572
NU_Remove_PPP_User .....	2574
NU_Set_PPP_Client_IP_Address .....	2575
NU_PPP_Still_Connected .....	2577
NU_Wait_For_PPP_Client .....	2579
NU_PPP_Get_Default_Options .....	2581
NU_Validate_Link_Options .....	2582
NU_Get_PPP_Link_Options .....	2584
NU_Get_PPP_Link_Option .....	2586
NU_Set_PPP_Link_Options .....	2589
NU_Set_PPP_Link_Option .....	2591
NU_PPP_Last_Activity_Time .....	2594
NU_PPP_Negotiation_Timeout .....	2595
NU_Obtain_PPP_Connection_Status .....	2596
NU_Abort_Wait_For_PPP_Client .....	2598
NU_PPP_Abort .....	2600
NU_PPP_Abort_Connection .....	2602
PPP Metadata Options .....	2603
PPP Configuration Options .....	2605
Link Negotiations .....	2609
Using PPP with SNMP .....	2609
IPv6 Over PPP .....	2610
MPPE Over PPP .....	2611
Control Compression Protocol Over PPP .....	2611
Multilink Protocol .....	2613
MP Services .....	2614
NU_MP_Get_Virt_If_By_Device .....	2615
NU_MP_Get_Virt_If_By_User .....	2616
NU_MP_Terminate_Links .....	2617
NU_MP_Get_Opt .....	2618
MP Initialization .....	2621
Example: MP Client .....	2621
Example: MP Server .....	2622
HDLC Interface .....	2624



---

HDLC Introduction .....	2624
HDLC Services .....	2624
NU_Reset_Modem .....	2626
NU_Carrier .....	2627
NU_Change_Communication_Mode .....	2629
NU_Modem_Control_String .....	2630
NU_Modem_Rings_To_Answer .....	2631
NU_Terminal_Data_Ready .....	2632
NU_Get_Terminal_Char .....	2633
NU_Purge_Terminal_Buffer .....	2635
NU_Put_Terminal_Char .....	2636
NU_Modem_Get_Remote_Num .....	2637
NU_Modem_Set_Local_Num .....	2638
HDLC Device Initialization .....	2638
HDLC Metadata Options .....	2639
HDLC Configuration Options .....	2640
HDLC_FOREIGN_DEFAULT_ACCM and HDLC_LOCAL_DEFAULT_ACCM ....	2640
MDM_RINGS_TO_ANSWER_ON .....	2641
HDLC_MAX_HOLDING_PACKETS and HDLC_MAX_HOLDING_PACKETS_PTR	2641
HDLC_MAX_TX_QUEUE_PTRS .....	2641
The Direct Cable Interface .....	2644
Setting up Direct Cable Connection for Windows .....	2644
Examples .....	2645
DC Device Initialization .....	2645
PPPoE Interface .....	2646
PPPoE Introduction .....	2646
PPPoE Services .....	2646
PPPoE Device Initialization .....	2646
PPPoE Configuration Options .....	2647
Macros and Definitions .....	2647
Data Structures .....	2648
Globals .....	2648
PPEC Cookie Functions .....	2649
PPEC_Encode_Cookie .....	2650
PPEC_Decode_Cookie .....	2652
Nucleus PPP Constants .....	2653
Nucleus PPP Constants (Alphabetical Listing) .....	2653
Additional Demonstrations .....	2654

## Appendix A

### Installing Certificates on Microsoft Windows ..... 2701

Installing Certificates Procedure .....	2701
Generating Certificates Using OpenSSL .....	2703

## Appendix B

### Installing Racoon2 ..... 2705

Installing Racoon2 Procedure .....	2705
Racoon2 Configuration .....	2705

---

Required Configuration Changes for Use with Nucleus IPsec .....	2706
Changes Required in racoon2.conf .....	2706
Changes Required in vals.conf .....	2707
Changes Required in default.conf .....	2707
Changes Required in transport_ike.conf .....	2707
Racoon2 Daemon Usage .....	2708

## **Embedded Software and Hardware License Agreement**



























## List of Figures

---

Figure 3-1. Buffer Management .....	987
Figure 3-2. Event Dispatcher Loop. ....	994
Figure 3-3. Interface Management. ....	996
Figure 4-1. Example NAT Router .....	1018
Figure 7-1. IPsec Communication Example .....	1189
Figure 7-2. IP Packet Being Secured Using ESP. ....	1192
Figure 7-3. IP Packet Being Secured Using AH .....	1194
Figure 7-4. Nucleus IPsec Database .....	1197
Figure 8-1. IKEv1 Exchange .....	1324
Figure 8-2. IKEv2 Exchange .....	1325
Figure 8-3. Nucleus IPsec - IKE Components. ....	1327
Figure 8-4. IKE and IPsec Database Configuration. ....	1340
Figure 8-5. IKE Environment .....	1404
Figure 8-6. Phase One Negotiation .....	1426
Figure 8-7. Phase Two Negotiation .....	1427
Figure 9-1. SNMP Engine .....	1494
Figure 9-2. Nucleus SNMP/MIB Engines .....	1496
Figure 9-3. Serial Output Example One .....	1504
Figure 9-4. Serial Output Example Two. ....	1504
Figure 9-5. Serial Output Example Three .....	1505
Figure 9-6. Get. ....	1705
Figure 9-7. Get-Next .....	1706
Figure 9-8. Get-Bulk .....	1706
Figure 9-9. Set Version One .....	1707
Figure 9-10. Set Version Two .....	1708
Figure 9-11. VACM is AccessAllowed Process Flow. ....	1713
Figure 9-12. Notification Originator Table Relationships. ....	1715
Figure 9-13. Access Verification for SNMPv1 .....	1720
Figure 9-14. Access Verification for SNMPv2c .....	1754
Figure 15-1. SMTP Overview .....	2405
Figure 15-2. SMTP_SESSION .....	2408
Figure 15-3. SMTP_MSG Object .....	2409
Figure 15-4. High-Level API Control Flow .....	2410
Figure 15-5. Session Connect Sequence .....	2412
Figure 15-6. Send Message Sequence .....	2413
Figure 17-1. JSON Parser Core .....	2510
Figure 17-2. Parsing a String. ....	2511
Figure 17-3. Stack Levels .....	2512
Figure 18-1. WPA Supplicant with Nucleus Products. ....	2539
Figure 18-2. Nucleus WPA Supplicant Configuration Files .....	2542

---

Figure 19-1. PPP Relationship with Other Modules and Other Nucleus Products . . . . .	2559
Figure 19-2. A Multilink PPP Connection. . . . .	2613
Figure 19-3. Relationship Between Components Involved in a Multilink Connection. . . . .	2613
Figure 19-4. HDLC_MAX_HOLDING_PACKETS and HDLC_MAX_HOLDING_PACKETS_PTR . . . . .	2642
Figure 19-5. HDLC_MAX_TX_QUEUE_PTRS . . . . .	2643
Figure A-1. Add Standalone Snap-in. . . . .	2702



## List of Tables

---

Table 1-1. Nucleus Networking Components .....	52
Table 2-1. DHCP6_IA_ADDR_STRUCT .....	57
Table 2-2. DEV6_PRFX_ENTRY .....	58
Table 2-3. ARP_ENTRY .....	59
Table 2-4. NU_BOOTP_STRUCT .....	59
Table 2-5. NU_DEVICE .....	60
Table 2-6. dev_flags Description .....	61
Table 2-7. IP6_POLICY_ENTRY .....	62
Table 2-8. DHCP_DUID_STRUCT .....	63
Table 2-9. NU_HOSTENT .....	64
Table 2-10. SCK_IOCTL_OPTION .....	65
Table 2-11. DEV6_RTR_OPTS .....	66
Table 2-12. NU_DHCP_STRUCT .....	67
Table 2-13. DHCP6_CLIENT .....	68
Table 2-14. RIP2_STRUCT .....	69
Table 2-15. RIPNG_STRUCT .....	70
Table 2-16. NU_DNS_SD_SERVICE .....	70
Table 2-17. NU_DNS_SD_INSTANCE .....	71
Table 2-18. UM_USER .....	72
Table 2-19. UPDATED_ROUTE_NODE .....	73
Table 2-20. cmsghdr .....	74
Table 2-21. msghdr .....	75
Table 2-22. in6_pktinfo .....	75
Table 2-23. addr_struct .....	76
Table 2-24. ip6_hbh .....	77
Table 2-25. ip6_rthdr .....	77
Table 2-26. sck_linger_struct .....	78
Table 2-27. if_nameindex .....	78
Table 2-28. Default Policy Table Parameters .....	140
Table 2-29. NU_DHCP_STRUCT Application Layer Members .....	162
Table 2-30. NU_DHCP_STRUCT Stack Members .....	162
Table 2-31. NU_Fcntl Supported Arguments .....	185
Table 2-32. SOL_SOCKET Level Options .....	235
Table 2-33. IPPROTO_IP Level Options .....	236
Table 2-34. IPPROTO_TCP Level Options .....	237
Table 2-35. IPPROTO_UDP Level Options .....	238
Table 2-36. IPPROTO_IPV6 Level .....	239
Table 2-37. NU_Ioctl Options .....	384
Table 2-38. NU_TYPE_STREAM or NU_TYPE_DGRAM Type .....	504
Table 2-39. NU_TYPE_RAW Type .....	504

---

Table 2-40. SOL_SOCKET Level Options .....	507
Table 2-41. IPPROTO_IP Level Options .....	507
Table 2-42. IPPROTO_IPv6 Level Options .....	509
Table 2-43. IPPROTO_TCP Level Options .....	511
Table 2-44. IPPROTO_UDP Level Options .....	517
Table 2-45. Using the Optimized Block Copy Routine .....	698
Table 3-1. Calls to be Modified for IPv6 .....	978
Table 3-2. Adding an IPv4 Address to a Device .....	982
Table 3-3. Adding an IPv6 Address to a Device .....	983
Table 3-4. IOCTL Actions .....	984
Table 3-5. Nucleus Threads .....	985
Table 3-6. Events Description .....	992
Table 3-7. Hardware Checksum Offloading Options .....	998
Table 3-8. NET Configuration Options .....	1000
Table 4-1. NAT Protocol .....	1019
Table 4-2. NAT Files .....	1020
Table 4-3. NAT_PORTMAP_SERVICE .....	1022
Table 5-1. DHCP Configuration Control Block .....	1051
Table 5-2. Required Control Block Options .....	1053
Table 5-3. DHCP_CLIENT_ID .....	1056
Table 6-1. SSL Packet .....	1179
Table 6-2. SSL Protocol Types .....	1179
Table 7-1. Supported IPsec Algorithms .....	1191
Table 7-2. ESP Header .....	1193
Table 7-3. ESP Trailer .....	1193
Table 7-4. AH Header .....	1194
Table 7-5. IPSEC_POLICY .....	1198
Table 7-6. IPSEC_SECURITY_PROTOCOL .....	1201
Table 7-7. IPSEC_SELECTOR .....	1203
Table 7-8. IPSEC_IP_ADDR .....	1205
Table 7-9. IPSEC_BUNDLE_LIST .....	1206
Table 7-10. IPSEC_SA_LIFETIME .....	1206
Table 7-11. IPSEC_OUTBOUND_SA .....	1207
Table 7-12. IPSEC_OUTBOUND_INDEX .....	1208
Table 7-13. IPSEC_INBOUND_SA .....	1209
Table 7-14. IPSEC_INBOUND_INDEX .....	1211
Table 7-15. Authentication Algorithms .....	1269
Table 7-16. Encryption Algorithms .....	1270
Table 8-1. IKE Policy List .....	1341
Table 8-2. IKE_POLICY .....	1343
Table 8-3. IKE_POLICY_SELECTOR/IKE_IDENTIFIER .....	1348
Table 8-4. IKE_SELECTOR_IP .....	1350
Table 8-5. IKE_IPS_ID .....	1351
Table 8-6. IKE_ATTRIB .....	1352
Table 8-7. IKE_SA_LIFETIME .....	1356

---

Table 8-8. IKE_KEY_PAIR .....	1356
Table 8-9. IKE_PRESHARED_KEY .....	1357
Table 8-10. IKE_INITIATE_REQ .....	1358
Table 8-11. Identification Methods .....	1359
Table 8-12. Pre-shared Keys .....	1359
Table 8-13. IKE_Get_Group_Opt .....	1377
Table 8-14. IKE_Get_Policy_Opt .....	1382
Table 8-15. IKE_Set_Policy_Opt .....	1400
Table 8-16. Bypass Security Policies .....	1404
Table 8-17. Network Traffic Security Policy .....	1405
Table 8-18. IKE Policy .....	1405
Table 8-19. IKE_Encryption_Algos .....	1416
Table 8-20. IKE_HASH_Algos .....	1419
Table 8-21. IKE Exchange .....	1428
Table 8-22. IKE Error Codes .....	1429
Table 9-1. MIBII_SERVICES .....	1497
Table 9-2. SNMP_PARITY Parameters .....	1503
Table 9-3. mib_element_t .....	1506
Table 9-4. CBSM_COMMUNITY_STRUCT .....	1507
Table 9-5. SNMP_CBSM_COMMUNITY_STRUCT .....	1509
Table 9-6. SNMP_NOTIFY_REQ_STRUCT .....	1510

---

Table 9-7. SNMP_NOTIFY_TABLE .....	1511
Table 9-8. SNMP_NOTIFY_FILTER_PROFILE_TABLE .....	1513
Table 9-9. SNMP_NOTIFY_FILTER_TABLE .....	1514
Table 9-10. SNMP_SS_STRUCT .....	1517
Table 9-11. SNMP_TARGET_ADDRESS_TABLE .....	1518
Table 9-12. SNMP_TARGET_PARAMS_TABLE .....	1520
Table 9-13. SNMP_USM_USER_STRUCT .....	1521
Table 9-14. USM_USERS_STRUCT .....	1523
Table 9-15. USM_PRIV_PROT_STRUCT .....	1525
Table 9-16. USM_AUTH_PROT_STRUCT .....	1525
Table 9-17. VACM_CONTEXT_STRUCT .....	1526
Table 9-18. VACM_SEC2GROUP .....	1527
Table 9-19. VACM_SEC2GROUP_STRUCT .....	1528
Table 9-20. VACM_ACCESS .....	1529
Table 9-21. VACM_ACCESS_STRUCT .....	1531
Table 9-22. VACM_VIEWTREE .....	1533
Table 9-23. VACM_VIEWTREE_STRUCT .....	1534
Table 9-24. System Group .....	1646
Table 9-25. Interfaces Group .....	1647
Table 9-26. Interface Extension Group .....	1649
Table 9-27. Interface Stack Group .....	1655
Table 9-28. Interface Receive Address Group .....	1655
Table 9-29. IP Group .....	1656
Table 9-30. ipRouteTable .....	1658
Table 9-31. ipNetToMediaTable .....	1660
Table 9-32. ICMP Group .....	1661
Table 9-33. TCP Group .....	1662
Table 9-34. UDP Group .....	1664
Table 9-35. EGP Group .....	1665
Table 9-36. SNMP Group .....	1666
Table 9-37. Nucleus SNMP Engine Group .....	1670
Table 9-38. Nucleus SNMP MPD Stats Group .....	1671
Table 9-39. Nucleus SNMP Target Address Table .....	1672
Table 9-40. Nucleus SNMP Target Params Table .....	1673
Table 9-41. Nucleus SNMP Notify Table .....	1674
Table 9-42. Nucleus SNMP Notify Filter Profile Table .....	1674
Table 9-43. Nucleus SNMP Notify Filter Table .....	1675
Table 9-44. Nucleus USM User Group .....	1675
Table 9-45. Nucleus Vacm Context Group .....	1677
Table 9-46. Nucleus Vacm Security to Group .....	1678
Table 9-47. Nucleus Vacm Access Group .....	1678
Table 9-48. Nucleus Vacm View Tree Family Group .....	1680
Table 9-49. Nucleus SNMP Community Table .....	1681
Table 9-50. Nucleus MIB Utility Options .....	1684
Table 9-51. PPP Link MIB .....	1687

---

Table 9-52. PPP Security MIB .....	1689
Table 9-53. PPP NCP MIB .....	1692
Table 9-54. Inputs to Access Control Subsystem .....	1710
Table 9-55. Outputs of Access Control Subsystem .....	1711
Table 10-1. FTP Commands .....	1989
Table 10-2. FTP Server and Client Return Codes .....	1991
Table 10-3. FTP Server and Client Internal Constants .....	1993
Table 10-4. IP_ADDR .....	1994
Table 10-5. FTP Server User-Configurable Constants .....	1994
Table 10-6. FTP Server Task-Related User-Configurable Variables .....	1995
Table 10-7. FTPSACCT .....	1996
Table 10-8. FTP Client User-Configurable Constants .....	2001
Table 10-9. FTP_CLIENT_STRUCT .....	2002
Table 10-10. NU_TN_PARAMETERS .....	2111
Table 10-11. Nucleus TFTP Server Defines .....	2156
Table 10-12. TFTP Options .....	2158
Table 10-13. NET_NTIFY_Debug_Struct .....	2163
Table 10-14. NET_NTIFY_STRUCT .....	2164
Table 10-15. Debug Messages .....	2165
Table 11-1. NU_HTTP_CLIENT .....	2271
Table 11-2. NU_HTTP_SSL_STRUCT .....	2272

---

Table 11-3. HTTP_SVR_SESSION_STRUCT .....	2274
Table 11-4. HTTP_TOKEN_INFO .....	2275
Table 12-1. WS_REQUEST .....	2303
Table 13-1. NU_WSOX_CONTEXT_STRUCTURE Members .....	2360
Table 14-1. SNTPC_TIME .....	2386
Table 14-2. SNTPC_SERVER .....	2387
Table 15-1. Sizes and Limits .....	2414
Table 15-2. Setting Options .....	2449
Table 16-1. JSON Generator Structure .....	2479
Table 16-2. JSON Generator Constants .....	2480
Table 17-1. JSON Parser Constants .....	2512
Table 17-2. JSON Parser Structure .....	2513
Table 17-3. JSON Parser String Structure .....	2513
Table 19-1. NU_PPP_OPTIONS .....	2560
Table 19-2. NU_PPP_CFG .....	2562
Table 19-3. use_flags Options for NU_PPP_CFG .....	2563
Table 19-4. NU_Get_PPP_Link_Option Options .....	2586
Table 19-5. NU_Set_PPP_Link_Option Options .....	2591
Table 19-6. NU_MP_Get_Opt Discriminator Class Options .....	2618
Table 19-7. NU_MP_Get_Opt PPP Options .....	2618
Table 19-8. Nucleus PPP Constants (Alphabetical Listing) .....	2653







# Chapter 1

## Nucleus Networking Components

---

The Nucleus networking components are provided to manage Nucleus networking applications. These components make up the networking package located at *os\networking*. Depending on your embedded application, any of these networking components can be configured for use.

---

### Note



Your source code is initially located in the `<install_root>\nucleus` directory. The source from this directory is copied into the project folder you specify.

---

This guide describes in detail each networking component and its APIs. For more information about packages and components, see “Nucleus ReadyStart Configuration” in the *Nucleus ReadyStart Guide* or “Nucleus Source Code Configuration” in the *Nucleus Source Code Guide*.

## Build Configurations

By default all components are enabled through the *.metadata* file located at the top level of each component’s installation, for example for WPA: `\os\networking\wpa_supplicant`. When a component is enabled, it is included in the build. You can create a user configuration file to override the default configuration and exclude a component from the build.

For example, if you have two different applications of Nucleus, one requiring IPsec, WPA, and WebServ and a second one requiring IPsec with IKE, SNTP, and WebServ, you can create two configuration files, one for each application.

For more information, see “Creating a Custom Configuration” in the *Nucleus ReadyStart Guide* or in the *Nucleus Source Code Guide*.

## Required Include File

To make all the networking package component APIs visible to an application, include the file *networking\nu\_networking.h* in your application, as in the following statement:

```
#include "networking/nu_networking.h"
```

---

### Warning



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## Networking Components

Table 1-1 summarizes each Nucleus networking component and links to additional detailed usage information in this document.

**Table 1-1. Nucleus Networking Components**

Networking Component	Description	Previously Documented in
<a href="#">NET</a>	Internet protocol services provided within Nucleus including support for IP4 and IP6.	Nucleus NET Reference Manual
<a href="#">Network Address Translator (NAT)</a>	IP Network Address Translator (NAT) router protocol that allows nodes on a private network to transparently communicate with nodes on an external network and vice versa.	Nucleus NAT Reference Manual
<a href="#">DHCP Server</a>	A DHCP server allocates network addressed and provides configuration parameters to dynamically configured clients.	New
<a href="#">Security Sockets Layer (SSL)</a>	Nucleus network cryptographic protocols: Transport Layer Security (TLS) and Secure Sockets Layer (SSL) through the mSSL toolkit.	Nucleus SSL User's Guide
<a href="#">IPsec</a>	Internet Protocol Security (IPsec) services and policy definitions supported by Nucleus.	Nucleus IPsec Reference Manual
<a href="#">IPsec-IKE</a>	Internet Key Exchange (IKE), a security protocol designed to automate management of Security Automation (SA).	Nucleus IPsec-IKE Supplement Reference Manual
<a href="#">Simple Network Management Protocol (SNMP)</a>	Nucleus SNMP is an embedded implementation of the management protocol as two separate products: Nucleus SNMP, supplies only version 1 functionality of the protocol and Nucleus SNMPv3, provides versions 1, 2c, and 3 agents.	Nucleus SNMP Reference Manual
<a href="#">Extended Protocol (XPROT)</a>	Nucleus Extended Protocol Services packaged under the XPROT name. This package contains an RFC compliant FTP Server, FTP Client, Telnet Server, Telnet Client and TFTP Server.	Nucleus Extended Protocol Package Reference Manual

**Table 1-1. Nucleus Networking Components (cont.)**

Networking Component	Description	Previously Documented in
<a href="#">WebServ</a>	HyperText Transfer Protocol (HTTP) server designed for embedded applications.	Nucleus WebServ Reference Manual
<a href="#">HTTP Lite</a>	Simple client/server implementation.	New
<a href="#">WebSocket</a>	The ReadyStart implementation of the WebSocket protocol complies with RFC 6455 to provide a complete WebSocket client/server implementation.	New
<a href="#">Simple Network Time Protocol (SNTP)</a>	Nucleus SNTP client is an embedded implementation of the SNTP Version 4 client tuned for applications where memory and CPU resources are limited.	Nucleus SNTP Reference Manual
<a href="#">Simple Mail Transfer Protocol (SMTP)</a>	SMTP protocol implementation, transfers E-Mail from one user on a network to the mailbox of another user.	New
<a href="#">JSON Generator</a>	The Nucleus JSON Generator allows you to create a JSON Document.	New
<a href="#">JSON Parser</a>	Nucleus provides a JSON Parser which can be used to parse JSON data.	New
<a href="#">Wireless Protected Access (WPA)</a>	Nucleus Wi-Fi Protected Access (WPA) security protocol for wireless computer networks, specifically the 802.11i and 802.1X IEEE standards.	WPA Supplicant User's Guide and Reference Manual
<a href="#">Point-To-Point Protocol (PPP)</a>	Nucleus PPP provides for communication between Nucleus embedded devices and hosts running Windows 95, Window NT, UNIX, and so on, connected via a point-to-point link.	Nucleus PPP Reference Manual

**Note**



The documents have been reorganized in the 2012.3 release. The manuals from prior releases have been grouped by category and relocated to a new document for that category to reduce the total number of documents and to increase navigability between them. The *Nucleus Networking Guide* is a new document that consists of reference manuals from prior releases that contain information pertaining to the networking components. The third column of [Table 1-1](#) maps earlier reference manuals to the chapters in the *Nucleus Networking Guide*.



## NET Overview

Nucleus OS integrates multiple networking services. This section describes APIs for Internet Protocol (IP) versions four (IPv4), six (IPv6), and APIs that apply to both protocols (IPv4/IPv6).

IPv4 is the fourth revision in the development of the Internet Protocol, and still by far is the most widely used Internet Layer Protocol, while IPv6 is in its beginning stages.

The following sections provide a detailed description of each API and caution about accessing the NET protocol outside the sockets interface.

All Nucleus NET access should be contained within the confines of the socket library. However, it is possible for calls to be made to the lower layers. Caution is in order when accessing the protocol stack without using the socket interface. This is due to the fact that all access to the protocol stack is semaphore protected to restrict access and preclude reentrancy failures.

---

### Warning



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

Most Nucleus NET services provide return values, which indicate the status of the request. See the following sections for the specific values returned for each of the interfaces. Note that beginning with release 4.3 of Nucleus NET the API functions may return Nucleus PLUS error codes. For example, a call to `NU_Socket()` might return `-32`, which is the value `NU_NO_MEMORY` if memory cannot be allocated for the socket to be created. Nucleus NET error codes are in the range of `-251` to `-500`.

Those Nucleus NET services with a '2' appended to the end of the name are new services created due to the addition of IPv6 functionality. These services replace an existing IPv4 service and should be used for IPv4 and IPv6 operations.

Those Nucleus NET services with a '6' appended to the end of the name are used only for IPv6 operations.

## Net Data Structures

This section describes data structures used by the [Net API Functions](#):

- [DHCP6\\_IA\\_ADDR\\_STRUCT](#)
- [DEV6\\_PRFX\\_ENTRY](#)
- [ARP\\_ENTRY](#)
- [NU\\_BOOTP\\_STRUCT](#)
- [NU\\_DEVICE](#)
- [IP6\\_POLICY\\_ENTRY](#)
- [DHCP\\_DUID\\_STRUCT](#)
- [NU\\_HOSTENT](#)
- [SCK\\_IOCTL\\_OPTION](#)
- [DEV6\\_RTR\\_OPTS](#)
- [NU\\_DHCP\\_STRUCT](#)
- [DHCP6\\_CLIENT](#)
- [RIP2\\_STRUCT](#)
- [RIPNG\\_STRUCT](#)
- [NU\\_DNS\\_SD\\_SERVICE](#)
- [NU\\_DNS\\_SD\\_INSTANCE](#)
- [UM\\_USER](#)
- [UPDATED\\_ROUTE\\_NODE](#)
- [cmsghdr](#)
- [msghdr](#)
- [in6\\_pktinfo](#)
- [addr\\_struct](#)
- [ip6\\_hbh](#)
- [ip6\\_rthdr](#)
- [sck\\_linger\\_struct](#)
- [if\\_nameindex](#)

## DHCP6\_IA\_ADDR\_STRUCT

DHCP6\_IA\_ADDR\_STRUCT is the typedef name for the dhcp6\_ia\_addr\_struct data structure which is defined as follows:

```
typedef struct dhcp6_ia_addr_struct
{
    UINT8    dhcp6_ia_addr[IP6_ADDR_LEN];
    UINT32    dhcp6_ia_pref_life;
    UINT32    dhcp6_ia_valid_life;
} DHCP6_IA_ADDR_STRUCT;
```

The members of the structure are defined in [Table 2-1](#):

**Table 2-1. DHCP6\_IA\_ADDR\_STRUCT**

Member	Description
dhcp6_ia_addr	The IPv6 address you want the DHCPv6 server to assign to the client. There is no guarantee that this address will actually be assigned to the client.
dhcp6_ia_pref_life	The preferred lifetime of the IPv6 address in the option, expressed in units of seconds. A value of 0 indicates that the client has no preference for this value.
dhcp6_ia_valid_life	The valid lifetime of the IPv6 address in the option, expressed in units of seconds. A value of 0 indicates that the client has no preference for this value.

## Related Topics

[Net Data Structures](#)

[DHCP6\\_Build\\_IA\\_NA\\_Option \(IPv6\)](#)

## DEV6\_PRFX\_ENTRY

The DEV6\_PRFX\_ENTRY is the typedef name for the struct\_dev\_prfx\_entry data structure which is defined as follows:

```
typedef struct_dev6_prfx_entry
{
    UINT8    prfx_prefix[16];
    INT      prfx_length;
    UINT32    prfx_adv_valid_lifetime;
    UINT32    prfx_adv_pref_lifetime;
    UINT32    prfx_flags;
} DEV6_PRFX_ENTRY;
```

Where the members of the data structure are defined in [Table 2-2](#) :

**Table 2-2. DEV6\_PRFX\_ENTRY**

Member	Description
prfx_prefix	The new prefix to add.
prfx_length	The length of the new prefix in bits.
prfx_adv_valid_lifetime	The valid lifetime of the new prefix in seconds, or 0 to use the default as specified by the <code>IP6_DEFAULT_ADV_VALID_LIFETIME</code> macro located in <i>networking/net6_cfg.h</i> .
prfx_adv_pref_lifetime	The preferred lifetime of the new prefix in seconds, or 0 to use the default as specified by the <code>IP6_DEFAULT_ADV_PREFERRED_LIFETIME</code> macro located in <i>networking/net6_cfg.h</i> .
prfx_flags	The flags to set for the prefix: <ul style="list-style-type: none"><li>• <code>PRFX6_NO_ADV_ON_LINK</code> - Do not set the on-link flag for this prefix in outgoing Router Advertisement messages.</li><li>• <code>PRFX6_NO_ADV_AUTO</code> - Do not set the autoconfig flag for this prefix in outgoing Router Advertisement messages.</li><li>• <code>PRFX6_DEC_VAL_LIFE</code> - Decrement the Valid Lifetime in real-time in outgoing Router Advertisement messages.</li><li>• <code>PRFX6_DEC_PREF_LIFE</code> - Decrement the Preferred Lifetime in real-time in outgoing Router Advertisement messages.</li></ul>

## Related Topics

[Net Data Structures](#)

[NU\\_Add\\_Prefix\\_Entry \(IPv6\)](#)

## ARP\_ENTRY

`ARP_ENTRY` is the typedef name for the `ARP_ENTRY_STRUCT` data structure, used for saving low level information. This ARP cache data structure is described below:

```
typedef struct ARP_ENTRY_STRUCT
{
    union {
        UINT8    ip_address[4];
        UINT32    arp_ip_addr;
    } ip_addr;

    INT32    arp_flags;
    UINT32    arp_time;
    INT32    arp_dev_index;
    UINT8    arp_mac_addr[DADDLEN];
} ARP_ENTRY;
```



The members of the structure are defined in [Table 2-3](#). Some of the members were omitted as they are used internally:

**Table 2-3. ARP\_ENTRY**

Member	Description
ip_addr	Union: the IP address.
arp_flags	Indicates if it is a gateway. For gateways an entry is created before the HW address is known. This flag indicates when the HW is resolved.
arp_time	Time information.
arp_dev_index	The device index.
arp_mac_addr	Hardware address for this IP address.

## Related Topics

[Net Data Structures](#)

[NU\\_ARP\\_Update](#)

## NU\_BOOTP\_STRUCT

NU\_BOOTP\_STRUCT is the typedef name for the bootp\_struct.

```
typedef struct bootp_struct
{
    UINT8 bp_ip_addr[4];    /* new IP address of client. */
    UINT8 bp_sname[64];    /* optional server host name field. */
    UINT8 bp_net_mask[4];  /* Net mask for new IP address */
    UINT8 bp_vend_opt[64]; /* Vendor-specific options */
} NU_BOOTP_STRUCT;
```

The members of the structure are defined in [Table 2-4](#). Some members of this structures have been omitted as they are used internally:

**Table 2-4. NU\_BOOTP\_STRUCT**

Member	Description
bp_ip_addr	If the client has already been configured with an IP address, this structure element should be filled in with the IP address.
bp_sname	If it is desired to only receive replies from one particular server, the client can use this element to specify the server to request.

**Table 2-4. NU\_BOOTP\_STRUCT (cont.)**

Member	Description
bp_net_mask	This is required if the BOOTP server is not configured to return a subnet mask. If the BOOTP server does return a subnet mask, and the application also specifies a subnet mask, the subnet mask from the server will be used.
bp_vend_opt	If it is desired that vendor-specific options be requested by the client, the bp_vend_opt field can be used. The application must ensure that the BOOTP magic cookie is placed into the field first, before any of the requested options are placed into the field.

## Related Topics

[Net Data Structures](#)

[NU\\_Bootp \(IPv4\)](#)

## NU\_DEVICE

NU\_DEVICE is the typedef name for the \_DEV\_DEVICE data structure which is defined as follows:

```
struct _DEV_DEVICE
{
    CHAR            *dv_name;
    UINT32          dv_flags;
    UINT32          dv_driver_options;
    STATUS          (*dv_init) (DV_DEVICE_ENTRY *);
    UINT8           dv_ip_addr[4];
    UINT8           dv_subnet_mask[4];
    UINT32          dv6_flags;
    DEV6_RTR_OPTS  dv6_rtr_opts;
    UINT32          dv_type;

    /* This union defines the hardware specific portion of the
     * device initialization structure.
     */
    union _dv_hw
    {
        URT_DEV      uart;
        ETHER_DEV     ether;
    } dv_hw;
};
```

The members of the structure are defined in [Table 2-5](#):

**Table 2-5. NU\_DEVICE**

Member	Description
dv_name	Pointer to the name of the interface.

**Table 2-5. NU\_DEVICE**

Member	Description
dv_flags	Flags to set for the interface. See <a href="#">Table 2-6</a> for a description of the available flags.
dv_driver_options	Advanced driver options used for interfaces other than Ethernet.
dv_init	A pointer to the initialization routine for the interface.
dv_ip_addr	The IPv4 address of the interface.
dv_subnet_mask	The IPv4 subnet mask of the interface.
dv6_flags	The IPv6 specific flags to set for the interface.
dv6_rtr_opts	The IPv6 router information to configure on the interface.
dv_type	If this is a VLAN interface the value DVT_VLAN should be used.
dv_hw.uart	If applicable, the UART information for the interface.
dv_hw.ether	If applicable, the IRQ and IO addr information for the interface.

[Table 2-6](#) contains a description of the dv\_flags:

**Table 2-6. dev\_flags Description**

Device Flag	Description
DV_VIRTUAL_DEV	This flag is used to denote a "virtual network device". This is usually used with PPP "Multilink" devices.
DV_POINTTOPOINT	This flag must be set on Point-to-Point (PPP) devices only. Please refer to the <a href="#">Point-To-Point Protocol (PPP)</a> chapter for more information related to this protocol.
DV_NOARP	Use this flag to disable the ARP protocol on the device. This flag is commonly used with Point-to-Point Protocol (PPP) devices.
DV_PROXYARP	Use this flag to enable "Proxy ARP" on the device. This can be used to proxy ARP replies for nodes that are unable to reply for themselves because of physically separate networks. It is commonly used with the Point-to-Point Protocol (PPP).
DV_CFG_IPV4_LL_ADDR	Enable IPv4 link-local address autoconfiguration on the interface.
DV6_IPV6	Use this flag to mark a device as being IPv6 enabled.
DV6_PRIMARY_INT	Marks the device is being the primary IPv6 device. Denotes the interface out of which a multicast packet should be transmitted when the multicast interface has not been set."
DV6_NODAD	Disable "Duplicate Address Detection" on an IPv6 device.

## Related Topics

[Net Data Structures](#)

[NU\\_Init\\_Devices \(IPv4/IPv6\)](#)

## IP6\_POLICY\_ENTRY

IP6\_POLICY\_ENTRY is the typedef name for the ip6\_policy\_entry data structure which is defined as follows:

```
typedef struct ip6_policy_entry
{
    UINT8      prefix[IP6_ADDR_LEN];
    UINT16     prefix_len;
    UINT8      precedence;
    UINT8      label;
} IP6_POLICY_ENTRY;
```

The members of the structure are defined in [Table 2-7](#):

**Table 2-7. IP6\_POLICY\_ENTRY**

Member	Description
prefix	The prefix to configure. This parameter must always be specified. This value cannot be changed once added. In order to change this value, you must delete the entry then add a new entry specifying the desired prefix.
prefix_len	The length of the prefix, in bits, of the prefix to configure. This parameter must always be specified. This value cannot be changed once added. In order to change this value, you must delete the entry then add a new entry specifying the desired prefix length.
precedence	The precedence of the prefix to configure. This value must always be specified when adding a new entry to the table and when updating an entry.
label	The label of the prefix to configure. This value must always be specified when adding a new entry to the table and when updating an entry.

## Related Topics

[Net Data Structures](#)

[NU\\_Configure\\_Policy\\_Table \(IPv6\)](#)

## DHCP\_DUID\_STRUCT

DHCP\_DUID\_STRUCT is the typedef name for the dhcp\_duid\_struct data structure which is defined as follows:

```
typedef struct dhcp_duid_struct
{
    UINT8    duid_id[DHCP_DUID_MAX_ID_NO_LEN];
    UINT8    duid_ll_addr[DADDLEN];
    UINT16   duid_id_no_len;
    UINT16   duid_type;
    UINT16   duid_hw_type;
    UINT32   duid_ent_no;
} DHCP_DUID_STRUCT;
```

The members of the structure are defined in [Table 2-8](#):

**Table 2-8. DHCP\_DUID\_STRUCT**

Member	Description
duid_id	The DUID (DHCP Unique Identifier).
duid_ll_addr	If the type of DUID is DUID-LL, this parameter holds the link-layer address associated with the DUID.
duid_id_no_len	The length of the DUID when using DUID_EN.
duid_type	The type of the DUID, either DHCP_DUID_LL or DHCP_DUID_EN.
duid_hw_type	If the type of DUID is DUID-LL, this parameter holds the hardware type associated with the DUID.
duid_ent_no	If the type of DUID is DUID-EN, this parameter holds the enterprise number associated with the DUID.

## Related Topics

[Net Data Structures](#)

[NU\\_Get\\_DHCP\\_DUID \(IPv4/IPv6\)](#)

[NU\\_Set\\_DHCP\\_DUID \(IPv4/IPv6\)](#)

## NU\_HOSTENT

NU\_HOSTENT is the typedef name for the NU\_Host\_Ent data structure which is defined as follows:

```
typedef struct NU_Host_Ent
{
    CHAR    *h_name;
    INT16   h_addrtype;
    INT16   h_length;
    CHAR    **h_addr_list;
} NU_HOSTENT;
```

The members of the structure are defined in [Table 2-9](#):

**Table 2-9. NU\_HOSTENT**

Member	Description
h_name	Pointer to the host name
h_addrtype	The family type of this host; either NU_FAMILY_IP for IPv4 or NU_FAMILY_IP6 for IPv6.
h_length	The valid lifetime of the IPv6 address in the option, expressed in units of seconds. A value of 0 indicates that the client has no preference for this value.
h_addr_list	A double pointer to the memory containing the list of IP addresses returned for this host.

## Related Topics

[Net Data Structures](#)

[NU\\_Delete\\_Host\\_Entry \(IPv4/IPv6\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

## SCK\_IOCTL\_OPTION

SCK\_IOCTL\_OPTION is the typedef name for the \_SCK\_IOCTL\_OPTION data structure which is defined as follows:

```
typedef struct _SCK_IOCTL_OPTION
{
    UINT8      *s_optval;
    struct
    {
        ARP_REQUEST arp_request;
        UINT8      s_ipaddr[MAX_ADDRESS_SIZE];
        UINT8      mac_address[DADDLEN];
        DV_REQ     s_dvreq;
        UINT16     vlan_id;
        UINT8      vlan_prio;
        UINT32     sck_bytes_pending;
    } s_ret;

    UINT8      s_optval_octet;
    UINT8      sck_max_msgs;
    UINT32     sck_interval;
} SCK_IOCTL_OPTION;
```

Where members of the structure are defined in [Table 2-10](#). Some members may have been omitted as they are used internally.

**Table 2-10. SCK\_IOCTL\_OPTION**

Member	Description
s_optval	Pointer to the data being forwarded.
s_ret.arp_request	ARP request data element.
s_ret.s_ipaddr	Field to store the IPv4 or IPv6 address.
s_ret.mac_address	Filed to store the MAC address.
s_ret.s_dvreq	Holds device request data.
s_ret.vlan_id	Used to store the VLAN id for an interface.
s_ret.vlan_prio	Used to store the VLAN priority for an interface.
s_ret.sck_bytes_pending	Returns the number of bytes pending to be read on a socket.
s_optval_octet	Optional variable.
sck_max_msgs	Maximum number of ICMP error messages in a certain interval.
sck_interval	Limit interval in which error messages will be transmitted.

## Related Topics

[Net Data Structures](#)

[NU\\_Ioctl\\_FIONREAD \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGETVLAN \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCDDARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFNETMASK \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGETVLANPRIO \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCIFREQ \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

## DEV6\_RTR\_OPTS

DEV6\_RTR\_OPTS is the typedef name for the `_dev6_rtr_opts` data structure which is defined as follows:

```
typedef struct _dev6_rtr_opts
{
    INT                rtr_MaxRtrAdvInterval;
```

```

        INT          rtr_MinRtrAdvInterval;
        INT32        rtr_AdvLinkMTU;
        INT32        rtr_AdvReachableTime;
    } DEV6_RTR_OPTS;

```

Where members of the structure are defined in [Table 2-11](#). Some members may have been omitted as they are used internally.

**Table 2-11. DEV6\_RTR\_OPTS**

Member	Description
rtr_MaxRtrAdvInterval	<p>The maximum time allowed between sending unsolicited multicast Router Advertisements from the interface, in seconds. MUST be no less than four seconds and no greater than 1800 seconds.</p> <ul style="list-style-type: none"> <li>-1 causes the default value to be used as specified by the <code>IP6_DEFAULT_MAX_RTR_ADVERT_INTERVAL</code> macro in <i>networking/net6_cfg.h</i>.</li> <li>-2 leaves the current value unmodified.</li> </ul>
rtr_MinRtrAdvInterval	<p>The minimum time allowed between sending unsolicited multicast Router Advertisements from the interface, in seconds. MUST be no less than three seconds and no greater than <math>.75 * \text{maxRtrInt}</math>.</p> <ul style="list-style-type: none"> <li>-1 causes the default value to be used as specified by the <code>IP6_DEFAULT_MIN_RTR_ADVERT_INTERVAL</code> macro in <i>networking/net6_cfg.h</i>.</li> <li>-2 leaves the current value unmodified.</li> </ul>
rtr_AdvLinkMTU	<p>The value to be placed in MTU options sent by the router. 0 indicates that no MTU options are sent.</p> <ul style="list-style-type: none"> <li>-1 causes the default value to be used as specified by the <code>IP6_DEFAULT_ADVERT_LINK_MTU</code> macro in <i>networking/net6_cfg.h</i>.</li> <li>-2 leaves the current value unmodified.</li> </ul>
rtr_AdvReachableTime	<p>The value to be placed in the Reachable Time field in the Router Advertisement messages sent by the router, in milliseconds. 0 means unspecified (by this router). MUST be no greater than 3,600,000 milliseconds (1 hour).</p> <ul style="list-style-type: none"> <li>-1 causes the default value to be used as specified by the <code>IP6_DEFAULT_ADVERT_REACHABLE_TIME</code> macro in <i>networking/net6_cfg.h</i>.</li> <li>-2 leaves the current value unmodified.</li> </ul>

## Related Topics

[Net Data Structures](#)

[NU\\_Configure\\_Router \(IPv6\)](#)

## NU\_DHCP\_STRUCT

NU\_DHCP\_STRUCT is defined as follows:



```
struct dhcp_struct
{
    UINT8    *dhcp_opts;
    UINT8    dhcp_opts_len;
    UINT8    dhcp_siaddr[4];
    UINT8    dhcp_giaddr[4];
    UINT8    dhcp_yiaddr[4];
    UINT8    dhcp_net_mask[4];
    UINT32    dhcp_xid;
    UINT8    dhcp_mac_addr[6];
    UINT8    dhcp_sname[64];
    UINT8    dhcp_file[128];
    UINT16    dhcp_secs;
};
```

The members of the structure are defined in [Table 2-12](#):

**Table 2-12. NU\_DHCP\_STRUCT**

Member	Description
dhcp_opts	A pointer to the options to be added to the Discover Packet. This is filled in by the Application Layer. Refer to RFC 2132 for the available options and the syntax for using the respective options.
dhcp_opts_len	Length, in octets, of the dhcp_opts field. This is filled in by the Application Layer.
dhcp_siaddr	The IP address of the DHCP Server which was queried.
dhcp_giaddr	The gateway IP address through which a default route was added to the Routing Table.
dhcp_yiaddr	The IP address assigned to the device by the DHCP Server.
dhcp_net_mask	The subnet mask of the IP address assigned by the DHCP Server.
dhcp_xid	The random transaction ID generated by the stack to use for the most recent DHCP Request issued by this device.
dhcp_mac_addr	The MAC (link-layer) address of the device.
dhcp_sname	The host name of the DHCP Server.
dhcp_file	Fully pathed file name returned by the DHCP Server.
dhcp_secs	Seconds since boot began.

## Related Topics

[Net Data Structures](#)

[NU\\_Dhcp \(IPv4\)](#)

[NU\\_Dhcp\\_Release \(IPv4\)](#)

## DHCP6\_CLIENT

DHCP6\_CLIENT is the typedef name for the dhcp6\_client data structure which is defined as follows:

```
typedef struct dhcp6_client
{
    UINT32    dhcp6_dev_index;
    UINT8     *dhcp6_user_opts;
    UINT16    dhcp6_opt_length;
} DHCP6_CLIENT;
```

The members of the structure are defined in [Table 2-13](#):

**Table 2-13. DHCP6\_CLIENT**

Member	Description
dhcp6_dev_index	The index of the interface on which to issue the DHCPv6 request.
dhcp6_user_opts	A pointer to the DHCPv6 options to include in the request. These options are built using the DHCP6_* API routines defined in this document. When requesting an IPv6 address to be assigned to the interface, the DHCP6_Build_IA_NA_Option API routine is used to build an IA NA option, to be transmitted to the DHCPv6 server. When requesting information only, no IA NA option should be specified, but only an Information Request Option containing the options being requested. You can request information and an IPv6 address in the same function call by including all required options.
dhcp6_opt_length	The length of the options in dhcp_user_opts.

## Related Topics

[Net Data Structures](#)

[NU\\_Dhcp6 \(IPv6\)](#)

## RIP2\_STRUCT

RIP2\_STRUCT is the typedef name for the \_RIP2\_STRUCT data structure which is defined as follows:

```
typedef struct _RIP2_STRUCT
{
    INT8          *rip2_device_name;
    UINT32        rip2_metric;
```

```

        INT8          rip2_sendmode;
        INT8          rip2_recvmode;
    } RIP2_STRUCT;

```

The members of the structure are defined in [Table 2-14](#):

**Table 2-14. RIP2\_STRUCT**

Member	Description
rip2_device_name	A pointer to the name of the interface on which to enable RIP.
rip2_metric	The metric to use for the interface. A value of zero will cause the metric of the interface to be used. The default value of the interface's metric is specified by the macro DEV_DEFAULT_METRIC in <i>networking/net_cfg.h</i> .
rip2_sendmode	The type of RIP packets to send from the interface.
rip2_recvmode	The type of incoming RIP packets to process on the interface: <ul style="list-style-type: none"> <li>• SEND_NONE Do not send RIP packets from this device.</li> <li>• SEND_RIP1 Send only IPv1 packets to the broadcast address from this device.</li> <li>• SEND_RIP2 Send only IPv2 packets to the broadcast address from this device.</li> <li>• SEND_RIP1_COMPAT Send only IPv2 packets to the broadcast address from this device.</li> <li>• RECV_NONE Do not receive RIP packets on this device.</li> <li>• RECV_RIP1 Process only IPv1 packets on this device.</li> <li>• RECV_RIP2 Process only IPv2 packets on this device.</li> <li>• RECV_RIP1_COMPAP Process both IPv1 and IPv2 packets on this device.</li> </ul>

## Related Topics

[Net Data Structures](#)

[NU\\_Rip2\\_Initialize \(IPv4\)](#)

## RIPNG\_STRUCT

RIPNG\_STRUCT is the typedef name for the \_RIPNG\_STRUCT data structure which is defined as follows:

```

typedef struct _RIPNG_STRUCT
{
    INT8          *ripng_device_name;
    UINT32        ripng_metric;
}

```

```
        STATUS          ripng_status;  
    } RIPNG_STRUCT;
```

The members of the structure are defined in [Table 2-15](#):

**Table 2-15. RIPNG\_STRUCT**

Member	Description
ripng_device_name	A pointer to the name of the device for which to enable RIPng. Note that the device must be IPv6-enabled by setting the DV6_IPV6 flag before the call to initialize the device.
ripng_metric	The metric to use for the device. A value of zero will cause the metric of the interface to be used. The default value of the interface's metric is specified by the macro DEV_DEFAULT_METRIC in <i>networking/net6_cfg.h</i> . The RIPng metric of a network is an integer between 1 and 15, inclusive. Given the maximum path limit of 15, a value of 1 is usually used.
ripng_status	The status of the attempt to initialize the interface for RIPng. If not NU_SUCCESS, either the interface is not IPv6-enabled (NU_INVALID_PARM) or the interface could not join the All-RIP-Routers multicast group (NU_INVAL).

## Related Topics

[Net Data Structures](#)

[NU\\_Ripng\\_Initialize \(IPv6\)](#)

## NU\_DNS\_SD\_SERVICE

NU\_DNS\_SD\_SERVICE is the typedef name for the \_nu\_dns\_sd\_service data structure which is defined as follows:

```
typedef struct _nu_dns_sd_service  
{  
    INT32  dns_key_len;  
    UINT16 dns_prio;  
    UINT16 dns_weight;  
    UINT16 dns_port;  
    CHAR   *dns_hostname;  
    CHAR   *dns_keys;  
} NU_DNS_SD_SERVICE;
```

The members of the structure are defined in [Table 2-16](#):

**Table 2-16. NU\_DNS\_SD\_SERVICE**

Member	Description
dns_key_len	The length of data in the *dns_keys.
dns_prio	Priority of the service.

**Table 2-16. NU\_DNS\_SD\_SERVICE**

Member	Description
dns_weight	Weight of the service.
dns_port	Port over which the service can be contacted.
dns_hostname	Hostname of the service that can be resolved to an IPv4 or IPv6 address.
dns_keys	Null-terminated keys related to the TXT record of the service. These keys can be parsed using NU_DNS_Parse_Key.

## Related Topics

[Net Data Structures](#)[NU\\_DNS\\_SD\\_Look\\_Up](#)

## NU\_DNS\_SD\_INSTANCE

NU\_DNS\_SD\_INSTANCE is the typedef name for the `_nu_dns_sd_instance` data structure which is defined as follows:

```
typedef struct _nu_dns_sd_instance
{
    struct _nu_dns_sd_instace  *dns_sd_next;
    CHAR                      *dns_sd_name;

} NU_DNS_SD_INSTANCE;
```

The members of the structure are defined in [Table 2-17](#):

**Table 2-17. NU\_DNS\_SD\_INSTANCE**

Member	Description
dns_sd_next	Pointer to the next instance.
dns_sd_name	Pointer to the name of the instance.

## Related Topics

[Net Data Structures](#)[NU\\_DNS\\_SD\\_Refresh](#)

## UM\_USER

UM\_USER data structure is defined as follows:

```
struct um_app_user_info
{
    CHAR    um_name[UM_MAX_NAME_SIZE + 1];
    CHAR    um_pw[UM_MAX_PW_SIZE + 1];
}
```

```
    UINT32 um_pv;  
    UINT32 um_id;  
};
```

The members of the structure are defined in [Table 2-18](#):

**Table 2-18. UM\_USER**

Member	Description
um_name	The name of the user.
um_pw	The password of the user.
um_pv	The registered services of the user: <ul style="list-style-type: none"><li>• UM_FTP – FTP service</li><li>• UM_PPP – PPP service</li><li>• UM_WEB – WebServer service</li></ul>
um_id	The unique identifier for this user's entry.

## Related Topics

[Net Data Structures](#)

[UM\\_Find\\_User](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Find\\_User\\_Next](#)

## UPDATED\_ROUTE\_NODE

UPDATED\_ROUTE\_NODE data structure is defined as follows:

```
struct updated_route_node  
{  
    UINT8    *urt_dest;  
    UINT8    *urt_gateway;  
    INT      urt_prefix_length;  
  
    union  
    {  
        CHAR    *urt_dev_name;  
        INT32    urt_dev_index;  
    }urt_dev;  
  
    INT32    urt_flags;  
  
    union  
    {  
        INT32    urt4_metric;  
        INT16    urt6_metric;  
    } urt_metric;  
  
    INT32    urt_age;  
    INT32    urt_routetag;  
    INT32    urt_path_mtu;  
};
```

The members of the structure are defined in [Table 2-19](#):

**Table 2-19. UPDATED\_ROUTE\_NODE**

Member	Description
urt_dest	The new destination address of the route or -1 to leave the destination address unchanged.
urt_gateway	Pointer to the new gateway (next-hop) of the route or -1 to leave the gateway unchanged.
urt_prefix_length	The new prefix length of the route or -1 to leave the prefix length unchanged.
urt_dev.urt_dev_name	The name of the new interface to use for the route or -1 to leave the interface unchanged. This is used for an IPv4 route only.
urt_dev.urt_dev_index	The interface index of the new interface to use for the route or -1 to leave the interface unchanged. This is used for an IPv6 route only.
urt_flags	Flags to set for the route or -1 to leave the flags unchanged: <ul style="list-style-type: none"> <li>• RT_UP – Flag the route as up.</li> <li>• RT_GATEWAY – The destination of the route is off-link.</li> <li>• RT_NOT_GATEWAY – The destination of the route is on-link.</li> <li>• RT_STATIC – The route is static and will not be deleted by routing protocols.</li> <li>• RT_SILENT – The route is silent and will not be advertised by routing protocols.</li> <li>• RT_LOCAL – The route was manually added.</li> <li>• RT_NETMGMT – The route was added via SNMP.</li> <li>• RT_RIP2 – The route was added via RIP-I or RIP-II discovery.</li> <li>• RT_RIPNG – The route was added via RIPng discovery.</li> <li>• RT_STOP_PMTU – The route does not perform Path MTU Discovery.</li> </ul>
urt_metric.urt4_metric	IPv4: The new metric of the route or -1 to leave the metric unchanged.
urt_metric.urt6_metric	IPv6: The new metric of the route or -1 to leave the metric unchanged.
urt_age	The new age of the route or -1 to leave the age unchanged.
urt_routetag	The new route tag of the route or -1 to leave the route tag unchanged.
urt_path_mtu	The new Path MTU of the route or -1 to leave the Path MTU unchanged.

## Related Topics

[Net Data Structures](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

## cmsghdr

Structure `cmsghdr` is defined as follows:

```
struct _cmsghdr
{
    UINT16  cmsg_len;
    INT     cmsg_level;
    INT     cmsg_type;
};
```

The members of the structure are defined in [Table 2-20](#):

**Table 2-20. cmsghdr**

Member	Description
<code>cmsg_len</code>	Number of bytes, including this header.
<code>cmsg_level</code>	Originating protocol.
<code>cmsg_type</code>	Protocol specific type.

## Related Topics

[Net Data Structures](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_CMSG\\_FIRSTHDR \(IPv6\)](#)

[NU\\_CMSG\\_NXTHDR \(IPv6\)](#)

## msghdr

Structure `_msghdr` is defined as follows:

```
struct _msghdr
{
    struct addr_struct *msg_name;
    UINT16             msg_namelen;
    CHAR               *msg_iov;
    UINT16             msg_iovlen;
    VOID               *msg_control;
    UINT16             msg_controllen;
    UINT16             flags;
};
```



The members of the structure are defined in [Table 2-21](#):

**Table 2-21. msghdr**

Member	Description
msg_name	Pointer to the address structure.
msg_namelen	Length of the address structure.
msg_iov	Pointer to the data buffer.
msg_iovlen	Length of the data buffer.
msg_control	Pointer to ancillary data buffer.
msg_controllen	Length of ancillary data buffer.
flags	Flags of received message .

## Related Topics

[Net Data Structures](#)

[NU\\_CMSG\\_NXTHDR \(IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

## in6\_pktinfo

in6\_pktinfo data structure is defined as follows:

```
struct _in6_pktinfo
{
    UINT8    ipi6_addr[16];
    UINT32   ipi6_ifindex;
};
```

The members of the structure are defined in [Table 2-22](#):

**Table 2-22. in6\_pktinfo**

Member	Description
ipi6_addr	The IPv6 address to set as the source address for outgoing packets sent using this socket. A value of all zeros will clear the previous source address set and allow the stack to select the source address..
ipi6_ifindex	The interface index of the interface to use for outgoing packets sent using this socket. A value of IP6_UNSPECIFIED will clear the previous interface index set and allow the stack to select the interface. Note that this does diverge from RFC 3542 which states a value of 0 should be specified. However, 0 is a valid interface index for Nucleus NET.

## Related Topics

[Net Data Structures](#)

[NU\\_Getsockopt\\_IPV6\\_PKTINFO \(IPv6\)](#)

[NU\\_Setsockopt\\_IPV6\\_PKTINFO \(IPv6\)](#)

## addr\_struct

addr\_struct data structure is defined as follows:

```
struct addr_struct
{
    INT16    family;
    UINT16   port;
    struct   id_struct id;
    char     *name;
};
```

The members of the structure are defined in [Table 2-23](#):

**Table 2-23. addr\_struct**

Member	Description
family	family = INTERNET
port	The port number of the machine.
id	Contains the 4-digit IP number for the host machine.
name	Points to the name of the machine.

## Related Topics

[Net Data Structures](#)

[NU\\_Accept \(IPv4/IPv6\)](#)

[NU\\_Bind \(IPv4/IPv6\)](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

[NU\\_Find\\_Socket \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_IPV6\\_NEXTHOP \(IPv6\)](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Send\\_To \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Setsockopt\\_IPV6\\_NEXTHOP \(IPv6\)](#)

[NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## ip6\_hbh

ip6\_hbh data structure (Hop-by-Hop options header) is defined as follows:

```
struct ip6_hbh
{
    UINT8    ip6h_nxt;
```

```
    UINT8    ip6h_len;
};
```

The members of the structure are defined in [Table 2-24](#):

**Table 2-24. ip6\_hbh**

Member	Description
ip6h_nxt	Next header.
ip6h_len	Length in units of 8 octets.

## Related Topics

[Net Data Structures](#)

[NU\\_Getsockopt\\_IPV6\\_HOPOPTS \(IPv6\)](#)

## ip6\_rthdr

ip6\_rthdr data structure (Routing header) is defined as follows:

```
struct ip6_rthdr
{
    UINT8    ip6r_nxt;
    UINT8    ip6r_len;
    UINT8    ip6r_type;
    UINT8    ip6r_segleft;
};
```

The members of the structure are defined in [Table 2-25](#):

**Table 2-25. ip6\_rthdr**

Member	Description
ip6r_nxt	Next header.
ip6r_len	Length in units of 8 octets.
ip6r_type	Routing type.
ip6r_segleft	Segments left.

## Related Topics

[Net Data Structures](#)

[NU\\_Getsockopt\\_IPV6\\_RTHDR \(IPv6\)](#)

[NU\\_Setsockopt\\_IPV6\\_RTHDR \(IPv6\)](#)

## sck\_linger\_struct

sck\_linger\_struct data structure (structure used for the linger socket option) is defined as follows:

```
struct sck_linger_struct
{
    INT          linger_on;
    UNSIGNED     linger_ticks;
};
```

The members of the structure are defined in [Table 2-26](#):

**Table 2-26. sck\_linger\_struct**

Member	Description
linger_on	Wait = NU_TRUE; Do not wait = NU_FALSE
linger_ticks	Wait in system ticks.

## Related Topics

[Net Data Structures](#)

[NU\\_Getsockopt\\_SO\\_LINGER \(IPv4/IPv6\)](#)

[NU\\_Setsockopt\\_SO\\_LINGER \(IPv4/IPv6\)](#)

## if\_nameindex

if\_nameindex data structure is defined as follows:

```
struct if_nameindex
{
    INT32      if_index;
    CHAR       *if_name;
};
```

The members of the structure are defined in [Table 2-27](#):

**Table 2-27. if\_nameindex**

Member	Description
if_index	Index: 1, 2, ....
if_name	Null terminated name: "le0", ....

## Related Topics

[Net Data Structures](#)

[NU\\_IF\\_FreeNameIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#)

## Net API Functions

This section describe the Nucleus NET API. Each section includes a function description, the function prototype, parameter descriptions, possible return values, and an example of usage for the interface. Each service also indicates whether it is to be used for IPv4-only, IPv6-only or IPv4 and IPv6 operations.

- [DHCP6\\_Build\\_IA\\_NA\\_Option \(IPv6\)](#)
- [DHCP6\\_Build\\_Option\\_Request\\_Option \(IPv6\)](#)
- [DHCP6\\_Build\\_User\\_Class\\_Option \(IPv6\)](#)
- [DHCP6\\_Build\\_Vendor\\_Class\\_Option \(IPv6\)](#)
- [DHCP6\\_Build\\_Vendor\\_Specific\\_Info\\_Option \(IPv6\)](#)
- [NETBOOT\\_Wait\\_For\\_Network\\_Up](#)
- [NU\\_Abort \(IPv4/IPv6\)](#)
- [NU\\_Accept \(IPv4/IPv6\)](#)
- [NU\\_Add\\_DNS\\_Server \(IPv4\)](#)
- [NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)
- [NU\\_Add\\_IP\\_To\\_Device \(IPv6\)](#)
- [NU\\_Add\\_Prefix\\_Entry \(IPv6\)](#)
- [NU\\_Add\\_Route \(IPv4\)](#)
- [NU\\_Add\\_Route6 \(IPv6\)](#)
- [NU\\_ARP\\_Update](#)
- [NU\\_Attach\\_IP\\_To\\_Device \(IPv4\)](#)
- [NU\\_Bind \(IPv4/IPv6\)](#)
- [NU\\_Bootp \(IPv4\)](#)
- [NU\\_Clear\\_Internal\\_Log](#)
- [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)

- [NU\\_CMSG\\_DATA \(IPv6\)](#)
- [NU\\_CMSG\\_FIRSTHDR \(IPv6\)](#)
- [NU\\_CMSG\\_LEN \(IPv6\)](#)
- [NU\\_CMSG\\_NXTHDR \(IPv6\)](#)
- [NU\\_CMSG\\_SPACE \(IPv6\)](#)
- [NU\\_Configure\\_IPv6\\_Interface \(IPv6\)](#)
- [NU\\_Configure\\_Policy\\_Table \(IPv6\)](#)
- [NU\\_Configure\\_Router \(IPv6\)](#)
- [NU\\_Connect \(IPv4/IPv6\)](#)
- [NU\\_Create\\_IPv4\\_Mapped\\_Addr \(IPv6\)](#)
- [NU\\_Delete\\_DNS\\_Server \(IPv4\)](#)
- [NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)
- [NU\\_Delete\\_Host\\_Entry \(IPv4/IPv6\)](#)
- [NU\\_Delete\\_Prefix\\_Entry \(IPv6\)](#)
- [NU\\_Delete\\_Route \(IPv4\)](#)
- [NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)
- [NU\\_Detach\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Device\\_Up \(IPv4/IPv6\)](#)
- [NU\\_Dhcp \(IPv4\)](#)
- [NU\\_Dhcp6 \(IPv6\)](#)
- [NU\\_Dhcp\\_Release \(IPv4\)](#)
- [NU\\_Dhcp6\\_Shutdown \(IPv6\)](#)
- [NU\\_Ethernet\\_Link\\_Down](#)
- [NU\\_DNS\\_SD\\_Browse](#)
- [NU\\_DNS\\_SD\\_Look\\_Up](#)
- [NU\\_DNS\\_SD\\_Refresh](#)
- [NU\\_DNS\\_Register\\_Service](#)
- [NU\\_DNS\\_Build\\_Key](#)
- [NU\\_DNS\\_Parse\\_Key](#)

- [NU\\_Ethernet\\_Link\\_Up](#)
- [NU\\_Fcntl \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Check \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Init \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Reset \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Set \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Next\\_Route\\_Entry](#)
- [NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Socket \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Default\\_Route \(IPv4\)](#)
- [NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Default\\_TTL \(IPv4\)](#)
- [NU\\_Get\\_DHCP\\_DUID \(IPv4/IPv6\)](#)
- [NU\\_Get\\_DHCP\\_IAID \(IPv4/IPv6\)](#)
- [NU\\_Get\\_DNS\\_Servers \(IPv4\)](#)
- [NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Domain\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)
- [NU\\_Get\\_Host\\_MX](#)
- [NU\\_Get\\_Host\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Forwarding \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Link\\_Local\\_Addr \(IPv6\)](#)
- [NU\\_Get\\_Peer\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_PMTU \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Reasm\\_Max\\_Size \(IPv4\)](#)

- `NU_Get_Sock_Name` (IPv4/IPv6)
- `NU_Getsockopt` (IPv4/IPv6)
- `NU_Getsockopt_IP_BROADCAST_IF` (IPv4)
- `NU_Getsockopt_IP_HDRINCL` (IPv4)
- `NU_Getsockopt_IP_MULTICAST_IF` (IPv4)
- `NU_Getsockopt_IP_MULTICAST_TTL` (IPv4)
- `NU_Getsockopt_IP_RECVIFADDR` (IPv4)
- `NU_Getsockopt_IP_TOS` (IPv4)
- `NU_Getsockopt_IP_TTL` (IPv4)
- `NU_Getsockopt_IPV6_CHECKSUM` (IPv6)
- `NU_Getsockopt_IPV6_DSTOPTS` (IPv6)
- `NU_Getsockopt_IPV6_HOPOPTS` (IPv6)
- `NU_Getsockopt_IPV6_MULTICAST_HOPS` (IPv6)
- `NU_Getsockopt_IPV6_MULTICAST_IF` (IPv6)
- `NU_Getsockopt_IPV6_NEXTHOP` (IPv6)
- `NU_Getsockopt_IPV6_PKTINFO` (IPv6)
- `NU_Getsockopt_IPV6_RECVDSTOPTS` (IPv6)
- `NU_Getsockopt_IPV6_RECVHOPLIMIT` (IPv6)
- `NU_Getsockopt_IPV6_RECVHOPOPTS` (IPv6)
- `NU_Getsockopt_IPV6_RECVPKTINFO` (IPv6)
- `NU_Getsockopt_IPV6_RECVRTHDR` (IPv6)
- `NU_Getsockopt_IPV6_RECVTCLASS` (IPv6)
- `NU_Getsockopt_IPV6_RTHDR` (IPv6)
- `NU_Getsockopt_IPV6_RTHDRDSTOPTS` (IPv6)
- `NU_Getsockopt_IPV6_TCLASS` (IPv6)
- `NU_Getsockopt_IPV6_UNICAST_HOPS` (IPv6)
- `NU_Getsockopt_IPV6_V6ONLY` (IPv6)
- `NU_Getsockopt_SO_BROADCAST` (IPv4)
- `NU_Getsockopt_SO_KEEPALIVE` (IPv4/IPv6)



- NU\_Getsockopt\_SO\_LINGER (IPv4/IPv6)
- NU\_Getsockopt\_SO\_RCVBUF (IPv4/IPv6)
- NU\_Getsockopt\_SO\_REUSEADDR (IPv4/IPv6)
- NU\_Getsockopt\_TCP\_CFG\_DSACK
- NU\_Getsockopt\_TCP\_CFG\_SACK
- NU\_Getsockopt\_TCP\_CONGESTION\_CTRL
- NU\_Getsockopt\_TCP\_DELAY\_ACK
- NU\_Getsockopt\_TCP\_FIRST\_RTO
- NU\_Getsockopt\_TCP\_FIRST\_TIMEOUT
- NU\_Getsockopt\_TCP\_KEEPALIVE\_R2
- NU\_Getsockopt\_TCP\_KEEPALIVE\_WAIT
- NU\_Getsockopt\_TCP\_MAX\_PROBES
- NU\_Getsockopt\_TCP\_MAX\_R2
- NU\_Getsockopt\_TCP\_MAX\_RTO
- NU\_Getsockopt\_TCP\_MAX\_SYN\_R2
- NU\_Getsockopt\_TCP\_MSL
- NU\_Getsockopt\_TCP\_NODELAY (IPv4/IPv6)
- NU\_Getsockopt\_TCP\_PROBE\_TIMEOUT
- NU\_Getsockopt\_TCP\_RCV\_WINDOWSIZE
- NU\_Getsockopt\_TCP\_SND\_WINDOWSIZE
- NU\_Getsockopt\_TCP\_TIMESTAMP
- NU\_Getsockopt\_TCP\_WINDOWSCALE
- NU\_Getsockopt\_UDP\_NOCHECKSUM (IPv4)
- NU\_IF\_FreeNameIndex (IPv4/IPv6)
- NU\_IF\_IndexToName (IPv4/IPv6)
- NU\_IF\_NameIndex (IPv4/IPv6)
- NU\_IF\_NameToIndex (IPv4/IPv6)
- NU\_Inet6\_Opt\_Append (IPv6)
- NU\_Inet6\_Opt\_Find (IPv6)

- [NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)
- [NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)
- [NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)
- [NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)
- [NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)
- [NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)
- [NU\\_Inet\\_NTOP \(IPv4/IPv6\)](#)
- [NU\\_Inet\\_PTON \(IPv4/IPv6\)](#)
- [NU\\_Init\\_Devices \(IPv4/IPv6\)](#)
- [NU\\_Init\\_Net \(IPv4/IPv6\)](#)
- [NU\\_Ioctl \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_FIONREAD \(IPV4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCDARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)
- [NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)
- [NU\\_Ioctl\\_SIOCGIFNETMASK \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGETVLAN \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGETVLANPRIO \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCICMPLIMIT](#)
- [NU\\_Ioctl\\_SIOCIFREQ \(IPv4/IPv6\)](#)

- [NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSETVLAN \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSETVLANPRIO \(IPv4/IPv6\)](#)
- [NU\\_IP\\_Multicast\\_Listen \(IPv4\)](#)
- [NU\\_IP6\\_Multicast\\_Listen \(IPv6\)](#)
- [NU\\_Is\\_Connected \(IPv4/IPv6\)](#)
- [NU\\_Is\\_IPv4\\_Mapped\\_Addr \(IPv6\)](#)
- [NU\\_Listen \(IPv4/IPv6\)](#)
- [NU\\_Ping \(IPv4\)](#)
- [NU\\_Ping2 \(IPv4/IPv6\)](#)
- [NU\\_Push \(IPv4/IPv6\)](#)
- [NU\\_Rarp \(IPv4\)](#)
- [NU\\_Recv \(IPv4/IPv6\)](#)
- [NU\\_Recvmsg \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_From \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_IF\\_Addr \(IPv4\)](#)
- [NU\\_Remove\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Remove\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Rip2\\_Initialize \(IPv4\)](#)
- [NU\\_Ripng\\_Initialize \(IPv6\)](#)
- [NU\\_Select \(IPv4/IPv6\)](#)
- [NU\\_Send \(IPv4/IPv6\)](#)
- [NU\\_Sendmsg \(IPv4/IPv6\)](#)
- [NU\\_Send\\_To \(IPv4/IPv6\)](#)
- [NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

- [NU\\_Set\\_Default\\_Hop\\_Limit \(IPv6\)](#)
- [NU\\_Set\\_Default\\_TTL \(IPv4\)](#)
- [NU\\_Set\\_Device\\_Hostname](#)
- [NU\\_Set\\_DHCP\\_DUID \(IPv4/IPv6\)](#)
- [NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)
- [NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Set\\_Host\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Set\\_IP\\_Forwarding \(IPv4/IPv6\)](#)
- [NU\\_Set\\_MDNS\\_Hostname\\_Callback](#)
- [NU\\_Set\\_Reasm\\_Max\\_Size \(IPv4\)](#)
- [NU\\_Shutdown \(IPv4/IPv6\)](#)
- [NU\\_Socket \(IPv4/IPv6\)](#)
- [NU\\_Setsockopt \(IPv4/IPv6\)](#)
- [NU\\_Setsockopt\\_IP\\_ADD\\_MEMBERSHIP \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_DROP\\_MEMBERSHIP \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_BROADCAST\\_IF \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_HDRINC \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_MULTICAST\\_IF \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_MULTICAST\\_TTL \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_RECVIFADDR \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_TOS \(IPv4\)](#)
- [NU\\_Setsockopt\\_IP\\_TTL \(IPv4\)](#)
- [NU\\_Setsockopt\\_IPV6\\_CHECKSUM \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_DSTOPTS \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_HOPOPTS \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_JOIN\\_GROUP \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_LEAVE\\_GROUP \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_MULTICAST\\_HOPS \(IPv6\)](#)
- [NU\\_Setsockopt\\_IPV6\\_MULTICAST\\_IF \(IPv6\)](#)

- `NU_Setsockopt_IPV6_NEXTHOP` (IPv6)
- `NU_Setsockopt_IPV6_PKTINFO` (IPv6)
- `NU_Setsockopt_IPV6_RECVDSTOPTS` (IPv6)
- `NU_Setsockopt_IPV6_RECVHOPLIMIT` (IPv6)
- `NU_Setsockopt_IPV6_RECVHOPOPTS` (IPv6)
- `NU_Setsockopt_IPV6_RECVPKTINFO` (IPv6)
- `NU_Setsockopt_IPV6_RECVRTHDR` (IPv6)
- `NU_Setsockopt_IPV6_RECVTCLASS` (IPv6)
- `NU_Setsockopt_IPV6_RTHDR` (IPv6)
- `NU_Setsockopt_IPV6_RTHDRDSTOPTS` (IPv6)
- `NU_Setsockopt_IPV6_TCLASS` (IPv6)
- `NU_Setsockopt_IPV6_UNICAST_HOPS` (IPv6)
- `NU_Setsockopt_IPV6_V6ONLY` (IPv6)
- `NU_Setsockopt_SO_BROADCAST` (IPv4)
- `NU_Setsockopt_SO_KEEPALIVE` (IPv4/IPv6)
- `NU_Setsockopt_SO_LINGER` (IPv4/IPv6)
- `NU_Setsockopt_SO_RCVBUF` (IPv4/IPv6)
- `NU_Setsockopt_SO_REUSEADDR` (IPv4/IPv6)
- `NU_Setsockopt_TCP_CFG_DSACK`
- `NU_Setsockopt_TCP_CFG_SACK`
- `NU_Setsockopt_TCP_CONGESTION_CTRL`
- `NU_Setsockopt_TCP_DELAY_ACK`
- `NU_Setsockopt_TCP_FIRST_PROBE_TIMEOUT`
- `NU_Setsockopt_TCP_FIRST_RTO`
- `NU_Setsockopt_TCP_KEEPALIVE_R2`
- `NU_Setsockopt_TCP_KEEPALIVE_WAIT`
- `NU_Setsockopt_TCP_MAX_PROBES`
- `NU_Setsockopt_TCP_MAX_R2`
- `NU_Setsockopt_TCP_MAX_RTO`

- [NU\\_Setsockopt\\_TCP\\_MAX\\_SYN\\_R2](#)
- [NU\\_Setsockopt\\_TCP\\_MSL](#)
- [NU\\_Setsockopt\\_TCP\\_PROBE\\_TIMEOUT](#)
- [NU\\_Setsockopt\\_TCP\\_NODELAY](#) (IPv4/IPv6)
- [NU\\_Setsockopt\\_TCP\\_RCV\\_WINDOWSIZE](#)
- [NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)
- [NU\\_Setsockopt\\_UDP\\_NOCHECKSUM](#) (IPv4)
- [NU\\_Setsockopt\\_TCP\\_WINDOWSCALE](#)
- [NU\\_Setup\\_Configured\\_Tunnel](#) (IPv6)
- [NU\\_Socket\\_Connected](#)
- [NU\\_Start\\_mDNS\\_Query](#)
- [NU\\_Stop\\_mDNS\\_Query](#)
- [NU\\_TFTPC\\_Get2](#) (IPv4/IPv6)
- [NU\\_TFTPC\\_Put2](#) (IPv4/IPv6)
- [NU\\_Test\\_Link\\_Up](#)
- [NU\\_Update\\_Route](#) (IPv4/IPv6)
- [NU\\_Update\\_DNS\\_Record](#)
- [NU\\_ZC\\_Allocate\\_Buffer](#) (IPv4/IPv6)
- [NU\\_ZC\\_BUF\\_LEN](#) (IPv4/IPv6)
- [NU\\_ZC\\_Deallocate\\_Buffer](#) (IPv4/IPv6)
- [NU\\_ZC\\_Recv](#) (IPv4/IPv6)
- [NU\\_ZC\\_Recv\\_From](#) (IPv4/IPv6)
- [NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL](#) (IPv4/IPv6)
- [NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT](#) (IPv4/IPv6)
- [NU\\_ZC\\_SEGMENT\\_DATA](#) (IPv4/IPv6)
- [NU\\_ZC\\_SEGMENT\\_NEXT](#) (IPv4/IPv6)
- [NU\\_ZC\\_Send](#) (IPv4/IPv6)
- [NU\\_ZC\\_Send\\_To](#) (IPv4/IPv6)

## DHCP6\_Build\_IA\_NA\_Option (IPv6)

This function builds an Identity Association for the Non-temporary Addresses Option (IA NA) in the provided buffer. The IA NA Option requests a DHCPv6 server to assign an IPv6 address to the client.

### Usage

```
UINT16 DHCP6_Build_IA_NA_Option (UINT8          *buffer,  
                                UINT32          identity_assoc,  
                                UINT32          time1,  
                                UINT32          time2,  
                                DHCP6_IA_ADDR_STRUCT *ia_addr_ptr,  
                                INT              ia_count);
```

### Arguments

- **buffer**  
A pointer to the buffer in which the option is built.
- **identity\_assoc**  
The identity association of the interface over which this DHCPv6 request is issued.
- **time1**  
The time at which the client contacts the server from which the addresses in the IA NA were obtained to extend the lifetimes of the addresses assigned to the IA NA; T1 is a time-duration relative to the current time expressed in units of seconds. The client sets T1 to 0 if it has no preference for those values; otherwise, this value indicates the client's preference.
- **time2**  
The time at which the client contacts any available server to extend the lifetimes of the addresses assigned to the IA NA; T2 is a time-duration relative to the current time expressed in units of seconds. The client sets T2 to 0 if it has no preference for those values; otherwise, this value indicates the client's preference.
- **ia\_addr\_ptr**  
A pointer to the buffer containing the IPv6 addresses to add to the IA NA option. If the client has no preference for which IPv6 address is assigned to the interface, set this value to NU\_NULL. The data structure [DHCP6\\_IA\\_ADDR\\_STRUCT](#) is described in the [Net Data Structures](#) section.
- **ia\_count**  
The number of IPv6 address in ia\_addr\_ptr.

### Return Values

- The number of bytes added to the user buffer.  
Upon completion, the IA NA Option is built in the user buffer.

## Example

```
UINT8          dhcp_buffer[256];
UINT32         iaid;
DHCP6_CLIENT   ds_ptr;
STATUS         status;

/* Get the index associated with this interface name. */
ds_ptr.dhcp6_dev_index = NU_IF_NameToIndex("eth0");

/* Get the IA ID associated with this interface. */
status = NU_Get_DHCP_IAID(ds_ptr.dhcp6_dev_index, &iaid);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Get_DHCP_IAID.\n");
}

/* Build an IA_NA option. */
ds_ptr.dhcp6_opt_length = DHCP6_Build_IA_NA_Option(dhcp_buffer,
                                                    iaaid, 0, 0, NU_NULL, 0);

. . .
```

## Related Topics

[Net API Functions](#)

[DHCP6\\_Build\\_User\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Specific\\_Info\\_Option \(IPv6\)](#)

[NU\\_Dhcp6 \(IPv6\)](#)

[NU\\_Dhcp6\\_Shutdown \(IPv6\)](#)

[NU\\_Get\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[DHCP6\\_Build\\_Option\\_Request\\_Option \(IPv6\)](#)



## DHCP6\_Build\_Option\_Request\_Option (IPv6)

This function builds an Option Request Option in the provided buffer. The Option Request Option is used to identify a list of options in a message between a client and a server.

### Usage

```
UINT16 DHCP6_Build_Option_Request_Option (UINT8    *buffer,
                                           UINT8    opt_count,
                                           UINT16   *opts_ptr);
```

### Arguments

- **buffer**  
A pointer to the buffer in which the option will be built.
- **opt\_count**  
The number of options being inserted in the Option Request Option.
- **opts\_ptr**  
A pointer to the options to add to the buffer.

### Return Values

- The number of bytes added to the user buffer.  
Upon completion, the Option Request Option is built in the user buffer.

### Example

```
UINT8      dhcp_buffer[256];
DHCP6_CLIENT ds_ptr;
UINT16     oro_ptr[2];

/* Fill in the options being requested. */
oro_ptr[0] = DHCP6_OPT_DNS_SERVERS;
oro_ptr[1] = DHCP6_OPT_DOMAIN_LIST;

/* Build a DNS Recursive Server Option and Domain Search List Option,
 * which are included in a single Option Request Option.
 */
ds_ptr.dhcp6_opt_length += DHCP6_Build_Option_Request_Option(dhcp_buffer,
                                                                2, oro_ptr);

. . .
```

## Related Topics

[Net API Functions](#)

[DHCP6\\_Build\\_Vendor\\_Class\\_Option \(IPv6\)](#)

[NU\\_Dhcp6 \(IPv6\)](#)

[NU\\_Get\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[DHCP6\\_Build\\_IA\\_NA\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_User\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Specific\\_Info\\_Option \(IPv6\)](#)

[NU\\_Dhcp6\\_Shutdown \(IPv6\)](#)

[NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

## DHCP6\_Build\_User\_Class\_Option (IPv6)

This function builds a User Class Option in the provided buffer. A client includes a User Class Option to identify the type or category of user or the applications it represents.

### Usage

```
UINT16 DHCP6_Build_User_Class_Option (UINT8 *buffer,  
                                       CHAR  *user_class_ptr,  
                                       INT    opt_count);
```

### Arguments

- **buffer**  
A pointer to the buffer in which the option will be built.
- **user\_class\_ptr**  
A pointer to the user-class-data information to add to the packet. Each user-class-data member is null-terminated to indicate the end of that entry.
- **opt\_count**  
The number of user-class-data entries in the user\_class\_ptr memory.

### Return Values

- The number of bytes added to the user buffer.  
Upon completion, the User Class Data Option is built in the user buffer.

### Example

```
UINT8  dhcp_buffer[256];  
UINT16 opt_length;  
CHAR   user_class[128];  
  
/* Copy the user-class-data member. */  
strcpy(user_class, "User number one\n");  
  
/* Build the User Class Data Option with one user-class-data member. */  
opt_length = DHCP6_Build_User_Class_Option(dhcp_buffer, user_class, 1);  
  
. . .
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>
<a href="#">NU_Dhcp6 (IPv6)</a>	<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>
<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## DHCP6\_Build\_Vendor\_Class\_Option (IPv6)

This function builds a Vendor Class Option in the provided buffer.

### Usage

```
UINT16 DHCP6_Build_Vendor_Class_Option (UINT8  *buffer,  
                                         UINT32  ent_number,  
                                         CHAR    *vendor_class_ptr,  
                                         INT     opt_count);
```

### Arguments

- **buffer**  
A pointer to the buffer in which the option will be built.
- **ent\_number**  
The enterprise number to insert in the option.
- **vendor\_class\_ptr**  
A pointer to the vendor-class-data to add to the option. Each vendor-class-data member is null-terminated to indicate the end of that entry.
- **opt\_count**  
The number of vendor-class-data entries in the vendor\_class\_ptr memory.

### Return Values

- The number of bytes added to the user buffer.  
Upon completion, the Vendor Class Option is built in the user buffer.

### Description

A client includes a Vendor Class Option to identify the vendor that manufactured the hardware on which the client is running.

### Example

```
UINT8  dhcp_buffer[256];  
UINT16 opt_length;  
CHAR    vendor_class[128];  
  
/* Copy the user-class-data member. */  
strcpy(vendor_class, "Vendor number one ");  
  
/* Build the User Class Option with one user-class-data member. */  
opt_length = DHCP6_Build_Vendor_Class_Option(dhcp_buffer, 0x98765432,  
                                              vendor_class, 1);  
  
. . .
```

## Related Topics

[Net API Functions](#)

[DHCP6\\_Build\\_Option\\_Request\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_User\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Specific\\_Info\\_Option \(IPv6\)](#)

[NU\\_Dhcp6 \(IPv6\)](#)

[NU\\_Dhcp6\\_Shutdown \(IPv6\)](#)

[NU\\_Get\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[DHCP6\\_Build\\_IA\\_NA\\_Option \(IPv6\)](#)

## DHCP6\_Build\_Vendor\_Specific\_Info\_Option (IPv6)

This function builds a Vendor Specific Information Option in the provided buffer. Clients and servers use the Vendor Specific Information Option to exchange vendor-specific information.

### Usage

```
UINT16 DHCP6_Build_Vendor_Specific_Info_Option (UINT8  *buffer,
                                                UINT32 ent_number,
                                                UINT16 opt_code,
                                                CHAR   *option_data);
```

### Arguments

- **buffer**  
A pointer to the buffer in which the option will be built.
- **ent\_number**  
The enterprise number to insert in the option.
- **opt\_code**  
The option code to add to the option-data opaque field.
- **option\_data**  
A pointer to the null-terminated option-data to add to the option-data opaque field.

### Return Values

- The number of bytes added to the user buffer.  
Upon completion, the Vendor Specific Information Option is built in the user buffer.

### Example

```
UINT8  dhcp_buffer[256];
UINT16 opt_length;
CHAR   option_data[64] = "Option data";

/* Build the Vendor Specific Information Option. */
opt_length = DHCP6_Build_Vendor_Specific_Info_Option(dhcp_buffer,
                                                    0x98765432, 12,
                                                    option_data);

. . .
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">NU_Dhcp6 (IPv6)</a>	<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>
<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	



## NETBOOT\_Wait\_For\_Network\_Up

Provides an API that the application can use to wait until the network stack is initialized.

### Usage

```
STATUS NETBOOT_Wait_For_Network_Up(UNSIGNED suspend);
```

### Arguments

- **suspend**  
Suspension option, indicating whether the routine will suspend pending full initialization of the networking stack and underlying transport drivers. Either `NU_SUSPEND` or `NU_NO_SUSPEND`.

### Return Values

- **NU\_SUCCESS**  
If successful.
- **NU\_TIMEOUT**  
If timeout on suspension.
- **NU\_NOT\_PRESENT**  
If event flags are present.

### Example

```
STATUS      status;

/* Wait until the networking stack is initialized. */
status = NETBOOT_Wait_For_Network_Up(NU_SUSPEND);

if (status == NU_SUCCESS)
{
    ...
}
else
{
    printf("NETBOOT_Wait_For_Network_Up() failed.\r\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Abort \(IPv4/IPv6\)](#)

## NU\_Abort (IPv4/IPv6)

This function is responsible for aborting a TCP or UDP connection.

### Usage

```
STATUS NU_Abort (INT socketd);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.

### Return Values

- `NU_SUCCESS`.  
Upon successful completion.  
Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_NO_PORT_NUMBER`  
No local port number was stored in the socket descriptor.

### Description

In the case of a TCP connection, a RESET is sent to the remote host. All resources for the connection are freed, and you do not need to call `NU_Close_Socket` to close the socket.

### Example

```
INT socketd; /* the original socket descriptor */  
  
status = NU_Abort(socketd);  
  
/* If status is NU_SUCCESS then the connection was  
   successfully aborted */
```

### Related Topics

[Net API Functions](#)

[NETBOOT\\_Wait\\_For\\_Network\\_Up](#)

## NU\_Accept (IPv4/IPv6)

This function is responsible for establishing a new socket descriptor containing information on both the server and the client.

### Usage

```
STATUS NU_Accept (INT socketd,  
                  struct addr_struct *peer,  
                  INT16 addrlen);
```

### Arguments

- **socketd**  
Specifies the server's socket descriptor.
- **peer**  
On return, this is a pointer to the protocol-specific address and family type of the client in the [addr\\_struct](#). The family parameter will hold the connection type; either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6. The port parameter will hold the foreign side's port number, and the id.is\_ip\_addrs parameter will hold the IP address of the foreign side of the connection.
- **addrlen**  
This parameter is reserved for future use. A value of zero should be used.

### Return Values

- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
- **NU\_INVALID\_PARM**  
The address of the client is invalid, if the sockets s\_flag is not set to listen, if the task\_entry is not valid, or the task\_entry ID is not valid.
- **NU\_NOT\_CONNECTED**  
The socket is not connected.
- **NU\_WOULD\_BLOCK**  
Socket is non-blocking but blocking is required to complete the requested action.
- **NU\_SOCKET\_CLOSED**  
The socket has been closed by another process.

### Description

Upon successful completion, the NU\_Accept routine returns a new socket descriptor with a value greater than or equal to zero. The new socket descriptor contains information specific to the server as well as the client. The original socket descriptor is maintained in its original

form in the event that another connection is established later between the same server and a different client. If a connection is not successfully established, the routine returns an operating system-specific error code or a Nucleus NET error code as described.

---

**Note**



The new socket descriptor inherits all application-set properties of the socket descriptor on which the connection was established. For example, if the application set the broadcast interface of the original socket via a call to NU\_Setsockopt, the new socket will inherit that broadcast interface.

---

It is only necessary for the server to call this function when a connection-oriented transfer is being established. NU\_Accept can be used in either blocking mode or non-blocking mode. In blocking mode, the calling task will be suspended until a connection is made. In non-blocking mode, the service returns immediately, even if no connections are ready to be accepted. Blocking is the default mode of operation. To make the service non-blocking, the NU\_Fcntl service must be used first. The new socket automatically inherits the blocking characteristic of the parent socket.

---

**Note**



Prior to the NU\_Accept service call, an application must call NU\_Listen. NU\_Listen sets up a table to accept connection requests. This table must be ready before NU\_Accept begins to block waiting for connection attempts.

---

### Example - IPv4 or IPv6

```
INT socketd;           /* the original socket descriptor */
INT new_socket;        /* the new socket descriptor      */
struct addr_struct peer; /* pointer to the client address structure*/

/* Create a socket for IPv4 or IPv6 connections */
socketd = NU_Socket(NU_FAMILY_IP6, NU_TYPE_STREAM, 0);

/* Server will accept connections from an IPv4 or IPv6 client */
new_socket = NU_Accept(socketd, &peer, 0);

/* If new_socket contains a value greater than or equal to 0, the
   NU_Accept call was successful. new_socket contains the socket
   descriptor, and peer points to the client address structure which
   contains the IP address and the family type of the client. */
```

### Example - IPv4

```
INT socketd;           /* the original socket descriptor */
INT new_socket;        /* the new socket descriptor      */
struct addr_struct peer; /* pointer to the client address structure*/

/* Create a socket for IPv4 connections only */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

/* Server will accept connections from an IPv4 client only */
new_socket = NU_Accept(socketd, &peer, 0);
```

```
/* If new_socket contains a value greater than or equal to 0, the  
   NU_Accept call was successful. new_socket contains the socket  
   descriptor, and peer points to the client address structure which  
   contains the IP address and the family type of the client. */
```

## Related Topics

[Net API Functions](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

[NU\\_Listen \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

[NU\\_Bind \(IPv4/IPv6\)](#)

## NU\_Add\_DNS\_Server (IPv4)

This function will add a new IPv4 DNS server to the list of IPv4 DNS servers that will be used when trying to resolve host addresses or host names.

### Usage

```
STATUS NU_Add_DNS_Server (UINT8 *new_dns_server,  
                          INT     where);
```

### Arguments

- `new_dns_server`  
This is a pointer to the address of an IPv4 DNS server to be added.
- `where`  
This is a flag that indicates where in the list the DNS server should be added. Valid values for this flag are `DNS_ADD_TO_FRONT` and `DNS_ADD_TO_END`. These will cause the DNS server to be added at the head or tail of the list respectively.

### Return Values

- `NU_SUCCESS`  
Upon successful completion, the new DNS server is added to the list. The new DNS server is available for use by any subsequent calls to [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#), or [NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#). Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_PARM`  
Either the server address is not a valid IP address or the where flag does not contain a valid value.
  - `NU_QUEUE_FULL`  
The maximum number of DNS servers has already been added. The configurable macro `DNS_MAX_DNS_SERVERS` located at *networking/net\_cfg.h* determines the maximum number of DNS servers.

### Description

The `NU_Get_IP_Node_By_Name` and `NU_Get_Host_By_Addr` functions make use of DNS servers to resolve names and addresses.

### Example

```
UINT8      dns1[] = {192,200,100,1};  
STATUS     status;  
  
/* Add a new DNS server to the head of the list. */  
status = NU_Add_DNS_Server(dns1, DNS_ADD_TO_FRONT);  
  
if (status != NU_SUCCESS)  
{
```

```
    printf("Error at call to NU_Add_DNS_Server.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Get\\_DNS\\_Servers \(IPv4\)](#)

[NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Delete\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

[NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)

## NU\_Add\_DNS\_Server2 (IPv4/IPv6)

This function is used for IPv4 and IPv6. It replaces the `NU_Add_DNS_Server` function. This function will add a new DNS server to the list of IPv4 or IPv6 DNS servers that will be used when trying to resolve host addresses or host names. The [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#) and [NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#) functions make use of DNS servers to resolve names and addresses.

### Usage

```
STATUS NU_Add_DNS_Server2 (UINT8 *new_dns_server,  
                           INT     where,  
                           INT16  family);
```

### Arguments

- `new_dns_server`

This is a pointer to the address of a DNS server to be added.

- `where`

This is a flag that indicates where in the list the DNS server should be added. Valid values for this flag are `DNS_ADD_TO_FRONT` and `DNS_ADD_TO_END`. These will cause the DNS server to be added at the head or tail of the list respectively.

- `family`

The family type of the DNS Server being added - either `NU_FAMILY_IP` for an IPv4 DNS Server or `NU_FAMILY_IP6` for an IPv6 DNS Server.

### Return Values

- `NU_SUCCESS`

Upon successful completion, the new DNS server will be added to the appropriate list. The new DNS server will be available for use by any subsequent calls to [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#) or [NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#). Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- `NU_INVALID_PARM`

The server address is not a valid IP address, the where flag does not contain a valid value, or the family specified is not supported.

- `NU_QUEUE_FULL`

The maximum number of DNS servers has already been added. The configurable macro `DNS_MAX_DNS_SERVERS` located at *networking/net\_cfg.h* determines the maximum number of DNS servers.

### Example - IPv4

```
UINT8      dns1[] = {192,200,100,1};  
STATUS     status;
```



```
/* Add a new IPv4 DNS server to the head of the list. */
status = NU_Add_DNS_Server2(dns1, DNS_ADD_TO_FRONT, NU_FAMILY_IP);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Add_DNS_Server2.\n");
}
```

### Example - IPv6

```
UINT8          dns1[] =
    {0xfe,0x80,0,0,0,0,0,0,0,0,0,0,0xa,0xb,0xff,0x2,0xc,0xd};
STATUS          status;

/* Add a new IPv6 DNS server to the head of the list. */
status = NU_Add_DNS_Server2(dns1, DNS_ADD_TO_FRONT, NU_FAMILY_IP6);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Add_DNS_Server2.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Get\\_DNS\\_Servers \(IPv4\)](#)

[NU\\_Add\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Delete\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

[NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)

## NU\_Add\_IP\_To\_Device (IPv6)

This function adds an IPv6 address to an IPv6 device.

### Usage

```
STATUS NU_Add_IP_To_Device (CHAR    *dev_name,  
                           UINT8    *ip_addr,  
                           UINT8    prefix_length,  
                           UINT32    preferred_lifetime,  
                           UINT32    valid_lifetime);
```

### Arguments

- **dev\_name**  
Pointer to the name of the IPv6-enabled device to which to add the IPv6 address.
- **ip\_addr**  
Pointer to the IPv6 address to add to the device.
- **prefix\_length**  
The length of the prefix in bits for the IPv6 address. If a prefix length is not specified, it will be calculated as the total IPv6 address length minus the length of the interface identifier for the device.
- **preferred\_lifetime**  
The amount of time the address should be considered Preferred. A value of 0xffffffff indicates an infinite Preferred Lifetime. Once the Preferred Lifetime expires, this address will not be used for newly established communications.
- **valid\_lifetime**  
The amount of time the address is valid. A value of 0xffffffff indicates an infinite valid lifetime. Once the valid lifetime expires, this address will be permanently deleted from the device.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
One of the input parameters is invalid.
  - **NU\_NO\_MEMORY**  
Insufficient memory in the system to create the new address structure.

## Description

IPv6 devices can have multiple IP addresses; therefore, you may add any number of IPv6 addresses to the device. The link-local address is added by default at the call to `NU_Init_Devices` and should never be manually added by the application. Also, Stateless Address Autoconfiguration will perform Router Solicitation on each device to obtain one or more globally unique IPv6 addresses. If an IPv6 router is configured on the network to assign IPv6 prefixes, when the call to `NU_Init_Devices` returns, the respective IPv6-enabled devices are assigned globally unique IPv6 addresses.

## Example

```
UINT8    ipv6_addr[] =
    {0x20, 2, 1, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1};

/* The device must be initialized before adding an IPv6 address */
NU_Init_Devices(DEMOI_Devices, 1);

/* Add the IPv6 address ipv6_addr with a prefix length of 64-
 * bits and a Preferred and Valid lifetime of infinity.
 */
NU_Add_IP_To_Device("eth0", ipv6_addr, 64,
    0xffffffff, 0xffffffff);
```

## Related Topics

[Net API Functions](#)

[NU\\_Remove\\_Device \(IPv4/IPv6\)](#)

[NU\\_Attach\\_IP\\_To\\_Device \(IPv4\)](#)

[NU\\_Detach\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)

[NU\\_Remove\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)

## NU\_Add\_Prefix\_Entry (IPv6)

This function adds a prefix entry to the list of prefixes for the specified interface.

### Usage

```
STATUS NU_Add_Prefix_Entry (UINT32          device_index,  
                           DEV6_PRFX_ENTRY *prefix_entry);
```

### Arguments

- `device_index`  
The index of the device on which to add the prefix entry.
- `prefix_entry`  
A pointer to the prefix entry to add. The data structure [DEV6\\_PRFX\\_ENTRY](#) is described in the [Net Data Structures](#) section.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_PARM`  
One of the input parameters is invalid.
  - `NU_NO_MEMORY`  
There is insufficient memory in the system to add the entry to the list.

### Description

If the interface specified has been flagged as a router, the new prefix will be advertised in Router Advertisements on the link. If `PRFX6_NO_ADV_ON_LINK` is not set in the `prfx_flags` member of the [DEV6\\_PRFX\\_ENTRY](#), a new IPv6 address will be assigned to the specified interface created from the prefix and the interface index of the specified interface.

### Example

```
UINT8 prefix[] = {0x3f, 0xfe, 0, 0, 1, 2, 0, 0};  
UINT32 device_index = 1;  
DEV6_PRFX_ENTRY    prefix_entry;  
  
/* Initialize the device and flag it as a Router */  
. . .  
  
/* Set up the members of the DEV6_PRFX_ENTRY data structure */  
memcpy(prefix_entry.prfx_prefix, prefix, 64 >> 3);  
prefix_entry.prfx_length = 64;  
prefix_entry.prfx_adv_valid_lifetime = 0;    /* Use default */  
prefix_entry.prfx_adv_pref_lifetime = 0;     /* Use default */  
prefix_entry.prfx_flags =  
    PRFX6_DEC_VAL_LIFE | PRFX6_DEC_PREF_LIFE;
```

```
/* Add the entry with a Prefix Length of 64, default Valid and
 * Preferred Lifetimes, and set the flags to decrement the
 * Preferred and Valid Lifetimes in Real-Time in outgoing
 * Router Advertisements sent from the interface.
 */
NU_Add_Prefix_Entry(device_index, &prefix_entry);
```

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Prefix\\_Entry \(IPv6\)](#)

## NU\_Add\_Route (IPv4)

This function is used by applications to add a route to the IPv4 routing table. This function can be used to add one default route as well as a normal route.

### Usage

```
STATUS NU_Add_Route (UINT8 *ip_dest,  
                    UINT8 *mask,  
                    UINT8 *gw_addr);
```

### Arguments

- **ip\_dest**  
This is the IP address of the destination to which the route is being added. There are three possible cases for this value. It can be a host IP address. It can be a network address. Or, it can be an IP address of all zeroes. The last case indicates that a default route is being added.
- **mask**  
This is a pointer to the subnet mask that should be used for this route.
- **gw\_addr**  
This is a pointer to the IP address of the gateway or next hop.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
One of the input parameters is invalid.

### Description

Note that a route is added automatically for each MAC layer device as it is initialized, including serial (SLIP and PPP) devices. So it is not necessary to add routes to directly connected hosts. There are three types of IPv4 routes that can be added. They are a host route, a network route, and a default route. Note that as of the NET 5.1 release, multiple routes to the same destination can exist as long as the gateway specified is different for each route.

A host route is only valid for a specific foreign host. The `ip_dest` parameter will be the address of the foreign host. The `mask` parameter will always be an address of 255.255.255.255 for a host route. The `gw_addr` will be the address of the next hop. When Nucleus NET is searching for a route to a destination, it always returns the most specific route. A host route is more specific than a network route, which is more specific than the default route. A match occurs when a bit-wise AND is performed between the destination address and the route mask, the result equals a

bit-wise AND of the destination stored in the route and the route mask. Because a host route has a mask of 255.255.255.255, only a specific destination IP address can match.

A network route specifies a route to a subnet of hosts. Thus a network route can match multiple IP addresses. For example, assume `ip_dest = 192.200.100.0`, `mask = 255.255.255.0`. Then the route would be valid for all IP addresses between 129.200.100.1 and 192.200.100.255. This is because when an AND is performed between those addresses and the mask, a match with 192.200.100.0 is realized.

The default route is used when all other options fail. The default route is very useful. Typically there will be only one or two routers on a network. One of these should be specified as the `gw_addr` for the default route. For the default route, `ip_dest` is always 0.0.0.0, `mask` is 0.0.0.0, and `gw_addr` is the IP address of the gateway to use as a default router. Note that there can be only one default route in the system at any time.

Since the purpose of a route is really to determine which device to use, a call to `NU_Init_Devices()` must be made to initialize devices before routes can be added.

### Example

```
UINT8 router[] = {200,100,100,1};

UINT8 subnet_mask[] = {255,255,255,0};
UINT8 netwrk[] = {192,200,100,0};

UINT8 mask[] = {255,255,255,255};
UINT8 host[] = {192,200,100,50};

UINT8 null_ip [] = {0,0,0,0};

/* Initialize Nucleus NET */
. . .

/* Initialize all devices */
. . .

/* Add a network route. */
NU_Add_Route(netwrk, subnet_mask, router);

/* Add a host route. */
NU_Add_Route(host, mask, router);

/* Add a default route. */
NU_Add_Route(null_ip, null_ip, router);
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

## NU\_Add\_Route6 (IPv6)

This function is used by applications to add a route to the IPv6 routing table.

### Usage

```
STATUS NU_Add_Route6 (UINT8 *ip_addr,  
                     UINT8 *gateway,  
                     UINT8 prefix_length);
```

### Arguments

- **ip\_addr**  
The destination IPv6 address of the route.
- **gateway**  
The IP address of the Next-Hop node.
- **prefix\_length**  
The length of the prefix in bits of the destination IP address. This value should always be 128 for host routes. If a prefix length is not specified, the default is 128.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_NO\_MEMORY**  
There is insufficient memory to add the route.
  - **NU\_INVALID\_PARM**  
A parameter passed in was NULL or there is not enough memory to complete the requested action.
  - **NU\_NO\_ACTION**  
A route to the destination through the specified gateway already exists.

### Description

Note that a route is added automatically for each MAC layer device as it is initialized.

Multiple routes to the same destination can exist as long as the gateway specified is different for each route.

Since the purpose of a route is really to determine which device to use, a call to `NU_Init_Devices()` must be made to initialize devices before routes can be added.

### Example - Host Route

```
UINT8 router[] = {0xfe,0x80,0,0,0,0,0,0,1,2,3,0xff,0xfe,4,5,6};  
UINT8 host[] = {0x3f,0xfe,1,2,3,4,0,0,0,0,0,0,0,0,5,6};
```



```
/* Initialize Nucleus NET */
. . .

/* Initialize all devices */
. . .

/* Add a Host route. */
NU_Add_Route6(host, router, 128);
```

### Example - Network Route

```
UINT8 router[] = {0xfe,0x80,0,0,0,0,0,0,1,2,3,0xff,0xfe,4,5,6};
UINT8 network[] = {0x3f,0xfe,0x29};

/* Initialize Nucleus NET */
. . .

/* Initialize all devices */
. . .

/* Add a Network route. */
NU_Add_Route6(network, router, 24);
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

## NU\_ARP\_Update

This function updates the parameters of the ARP Cache entry associated with the IP address.

### Usage

```
STATUS NU_ARP_Update(ARP_ENTRY *arp_changes,  
                    const UINT8 *target_ip);
```

### Arguments

- `arp_changes`  
A pointer to the data structure [ARP\\_ENTRY](#) that is holding the updated parameters. [ARP\\_ENTRY](#) is described in the [Net Data Structures](#) section.
- `target_ip`  
The IP address associated with the updated entry.

### Return Values

- `NU_SUCCESS`  
Indicates success.
- `NU_INVALID`  
Indicates that one of the parameters is invalid.
- `NU_NO_MEMORY`  
Indicates there is insufficient memory.

### Example

```
ARP_ENTRY    arp_entry;  
UINT8        addr[192,168,1,1];  
STATUS        status;  
  
/* Initialize all values in the arp_entry data structure to -1 */  
memset(&arp_entry, -1, sizeof(ARP_ENTRY));  
  
arp_entry.arp_flags = ARP_UP;  
  
/* Indicate that a different interface should be used to communicate with  
this host. */  
arp_entry.arp_dev_index = 1;  
  
/* Update the entry. */  
status = NU_ARP_Update(&arp_entry, addr);
```

### Related Topics

[Net API Functions](#)

[NU\\_Iocctl\\_SIOCGARP \(IPv4\)](#)

[NU\\_Iocctl\\_SIOCDARP \(IPv4\)](#)

## NU\_Attach\_IP\_To\_Device (IPv4)

This function adds an IP address to an interface. If the interface was previously disabled, this routine also re-enables the interface to transmit data.

### Usage

```
STATUS NU_Attach_IP_To_Device (CHAR    *name,  
                              UINT8   *ip_addr,  
                              UINT8   *subnet);
```

### Arguments

- **name**  
A pointer to the name of the interface.
- **ip\_addr**  
A pointer to the IP address to add to the interface.
- **subnet**  
A pointer to the subnet mask to use for the interface.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise the routine returns an operating system-specific error code, or a Nucleus NET error code.
  - **NU\_INVALID\_PARM**  
There is no interface in the system with the associated name, or one of the input parameters is NULL.

### Description

If you add an IP address in the range of the IPv4 link-local address space (169.254.1.0 -> 169.254.254.255), IPv4 link-local address configuration (per RFC 3927) will be initiated on the interface. If the requested address is proven to be in use by another node on the local link, the stack will generate a new random link-local address until a unique address is found. You can determine the address assigned to the interface via the API routine NU\_Ioctl(), specifying the option SIOCGIFADDR. If the API deletes the link-local address, IPv4 link-local address configuration will be disabled on the interface.

Note that an IPv4 link-local and globally routable address cannot be assigned to the interface at the same time. If you assign a globally routable address to an interface already configured with an IPv4 link-local address, the IPv4 link-local address will be removed, but IPv4 link-local configuration will not be disabled in anticipation of the globally routable address being deleted. Likewise, if you attempt to assign an IPv4 link-local address to an interface already configured with a globally routable address, the link-local address will not be assigned to the interface.

### Example – Re-enable an interface after disabling it

```
UINT8    ip_addr[4] = {192, 168, 1, 21};
UINT8    subnet[4] = {255, 255, 255, 0};

/* Disable the interface */
. . .

/* Re-enable the interface with a new IP address */
if (NU_Attach_IP_To_Device("Ethernet_0", ip_addr, subnet) != NU_SUCCESS)
{
    /* log an error */
    printf("Error at call to NU_Attach_IP_To_Device().\n");
}
```

### Example - Add an alias address to the interface

```
UINT8    ip_addr[4] = {192, 168, 10, 210};
UINT8    subnet[4] = {255, 255, 255, 0};

/* Initialize the device with a valid IP address or obtain an IP
 * address dynamically via DHCP, BOOTP or RARP
 */
. . .

/* Add an alias IP address to the interface */
if (NU_Attach_IP_To_Device("Ethernet_0", ip_addr, subnet) != NU_SUCCESS)
{
    /* log an error */
    printf("Error at call to NU_Attach_IP_To_Device().\n");
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Detach_IP_From_Device (IPv4/IPv6)</a>
<a href="#">NU_Device_Up (IPv4/IPv6)</a>	<a href="#">NU_Remove_Device (IPv4/IPv6)</a>
<a href="#">NU_Remove_IP_From_Device (IPv4/IPv6)</a>	<a href="#">NU_Attach_IP_To_Device (IPv4)</a>

## NU\_Bind (IPv4/IPv6)

This function is responsible for assigning a local address to a socket.

### Usage

```
STATUS NU_Bind (INT          socketd,  
                struct addr_struct *myaddr,  
                INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **myaddr**  
Pointer to the server's protocol-specific address in [addr\\_struct](#). You must fill in the family parameter with the family type of the local side of the connection; either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6, the port parameter with the port number of the local side of the connection, and the id.is\_ip\_addrs parameter with the IP address of the local side of the connection.
- **addrlen**  
This parameter is reserved for future use. A value of zero should be used.

### Return Values

- Upon successful completion, NU\_Bind() updates the local address portion of the socket descriptor and returns socketd. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
The address of the server is invalid.
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PORT**  
The port number does not equal 0 or the port number is not unique.
  - **NU\_INVALID\_ADDRESS**  
The address does not match a device listed.

### Description

A server must call it, whether a connection-oriented or connectionless transfer is being established. A client, however, only needs to call this function during a connectionless transfer.

### Example - IPv4

```
INT socketd;    /* the socket descriptor */

/* the server address structure*/
struct addr_struct *myaddr;

/* the user must fill in the address of the local machine before
   making the NU_Bind call */

myaddr->family = NU_FAMILY_IP;    /* Specifies IPv4 */
myaddr->port = 4000;              /* local machine's port num */
myaddr->id.is_ip_addrs[0] = 192;
myaddr->id.is_ip_addrs[1] = 9;    /*local machine's */
myaddr->id.is_ip_addrs[2] = 200;  /* 4-digit IP number*/
myaddr->id.is_ip_addrs[3] = 3;
myaddr->name = "server_name"

socketd = NU_Bind(socketd, myaddr, 0);
/* If socketd contains a value greater than or equal to 0, the
   NU_Bind call was successful and socketd contains the socket
   descriptor */
```

### Example - IPv6

```
INT socketd;    /* the socket descriptor */

/* the server address structure*/
struct addr_struct *myaddr;

/* the user must fill in the address of the local machine before
   making the NU_Bind call */

myaddr->family = NU_FAMILY_IP6;    /* Specifies IPv6 */
myaddr->port = 4000;              /* local machine's port num */
myaddr->id.is_ip_addrs[0] = 0xfe;
myaddr->id.is_ip_addrs[1] = 0x80;  /* local machine's */
myaddr->id.is_ip_addrs[2] = 0;     /* 16-digit IP number*/
myaddr->id.is_ip_addrs[3] = 0;
myaddr->id.is_ip_addrs[4] = 0;
myaddr->id.is_ip_addrs[5] = 0;
myaddr->id.is_ip_addrs[6] = 0;
myaddr->id.is_ip_addrs[7] = 0;
myaddr->id.is_ip_addrs[8] = 0;
myaddr->id.is_ip_addrs[9] = 0;
myaddr->id.is_ip_addrs[10] = 0xa;
myaddr->id.is_ip_addrs[11] = 0xb;
myaddr->id.is_ip_addrs[12] = 0xff;
myaddr->id.is_ip_addrs[13] = 0x2;
myaddr->id.is_ip_addrs[14] = 0xc;
myaddr->id.is_ip_addrs[15] = 0xd;
myaddr->name = "server_name"

socketd = NU_Bind(socketd, myaddr, 0);

/* If socketd contains a value greater than or equal to 0, the
   NU_Bind call was successful and socketd contains the socket
   descriptor */
```

## Example - Any IP Family

```
INT socketd;    /* the socket descriptor */

/* the server address structure*/
struct addr_struct *myaddr;

/* the user must fill in the address of the local machine before
   making the NU_Bind call */

myaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 or IPv4*/
myaddr->port = 4000;             /* local machine's port num */
myaddr->id = IP6_ADDR_ANY;
myaddr->name = "server_name"

socketd = NU_Bind(socketd, myaddr, 0);

/* If socketd contains a value greater than or equal to 0, the
   NU_Bind call was successful and socketd contains the socket
   descriptor */
```

## Related Topics

[Net API Functions](#)

[NU\\_Listen \(IPv4/IPv6\)](#)

[NU\\_Accept \(IPv4/IPv6\)](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

## NU\_Bootp (IPv4)

This function is used for IPv4 only. The Bootstrap Protocol provides a means for a diskless device to acquire an IPv4 address and configuration information.

### Usage

```
STATUS NU_Bootp (CHAR          *dv_name,  
                NU_BOOTP_STRUCT *bp_ptr);
```

### Arguments

- **dv\_name**  
Pointer to the name of the device for which BOOTP will be used to discover an IP address.
- **bp\_ptr**  
Pointer to a structure that is used to pass information to and get information from the NU\_Bootp service.

The [NU\\_BOOTP\\_STRUCT](#) data structure is defined in the [Net Data Structures](#) section. This structure is used to return configuration information to the application. This structure can be modified to hold vendor options that are returned in a BOOTP reply packet.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion of this service, the local host's adapter will have an IP address attached. The BOOTP server may have optionally returned a filename to the client. If so, this filename is passed up to the application through the NU\_BOOTP\_STRUCT structure. A file transfer protocol such as TFTP or FTP can be used to retrieve the file from the server. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_BOOTP\_INIT\_FAILED**  
A required resource could not be allocated.
  - **NU\_NO\_BUFFERS**  
A buffer to build the BOOTP request in could not be allocated.
  - **NU\_BOOTP\_SEND\_FAILED**  
BOOTP could not send the configuration request, possibly due to a driver problem.
  - **NU\_BOOTP\_ATTACH\_IP\_FAILED**  
BOOTP obtained an IP address but was unable to assign it to the network device.
  - **NU\_BOOTP\_FAILED**  
General BOOTP failure case.
  - **NU\_INVALID\_PARM**



One or more input parameters are invalid.

## Description

The Bootstrap Protocol is performed with a single packet exchange between the BOOTP client and the BOOTP Server. The Client sends the BOOTP Request. After receiving the BOOTP Request, the Server sends the BOOTP Reply filled with data that is configured within the BOOTP Server. The BOOTP Client receives the packet and parses out the IP address and any other configuration information provided in the response.

Optionally, the name of a remote file can be provided in the response. This file, which might contain configuration data not defined by the Bootstrap Protocol, can then be retrieved. Typically TFTP would be used to retrieve the remote file. The actual processing of the file is left up to the application.

The NU\_Bootp service provides the means by which Nucleus NET applications can access the Bootstrap Protocol. If required, this function should be called soon after power-up, but after setting up the device structures as shown in the example at the end of this section. By default support for BOOTP will not be included in the Nucleus NET library. The NU\_Bootp service is enabled by setting the define INCLUDE\_BOOTP in *networking/net\_cfg.h* to an NU\_TRUE. The default definition found in *networking/net\_cfg.h* is shown in the following example.

By default BOOTP client is not included in the Nucleus NET Build. To include it, change the following NU\_FALSE to a NU\_TRUE.

```
#define INCLUDE_BOOTP  NU_FALSE
```

NU\_Bootp can be used for multiple device interfaces to acquire IP addresses. The following example shows how two devices acquire IP addresses by using the NU\_Bootp service.

When the NU\_Bootp service call is used to acquire an IP address, the IP address that is passed to the NU\_Init\_Devices service must be set to all 0's.

The BOOTP Server must be configured to transmit Reply messages via broadcast, because Nucleus NET can only receive broadcast packets on interfaces that do not yet have an attached IP address. Some BOOTP servers default to sending the reply to the IP address that is being returned. Nucleus NET will reject such a reply.

If the BOOTP Server will be returning vendor options in the reply, you may have to configure the BOOTP Client to handle some of the vendor options. This can be accomplished by modifying the function BOOTP\_Process\_Packets in the file *networking/net/src/bootp.c* to handle the additional vendor options. Describing the modifications necessary to process vendor options, is beyond the scope of this document.

## Example

```
#define DEVICE_COUNT  2

NU_BOOTP_STRUCT      *bootp_ptr, *bp;
NU_DEVICE             devices[DEVICE_COUNT];
NU_MEMORY_POOL        DEMOI_NET_Memory_Pool;

extern char __NOCACHE_END[];
```

```

extern char __NOCACHE_START[];

void tcp_server_task(UNSIGNED argc, VOID *argv)
{
    char    c;
    int     socketd, newsock; /* the socket descriptor */
    struct addr_struct *servaddr; /* server address structure*/
    VOID     *pointer;
    STATUS   status;
    struct addr_struct client_addr;
    UINT32   ipaddr;
    INT16    ret;
    UINT32   ip_addr;
    char     subnet[] = {255,255,255,0};
    NU_NET_INIT_DATA init_struct;
    unsigned long nocacheSize = (((unsigned long)(__NOCACHE_END)) -
                                ((unsigned long)(__NOCACHE_START)));

    /* Create the memory pool for use by NET. It will contain the NET memory
       buffers. */
    status = NU_Create_Memory_Pool(&DEMOI_NET_Memory_Pool, "NETMEM",
                                (void *)__NOCACHE_START, nocacheSize,
                                8, NU_FIFO);

    /* Make sure the allocation worked. */
    if (status != NU_SUCCESS)
    {
        printf("Error at call to NU_Create_Memory_Pool().\n");
        NU_SUSPEND_Task(NU_Current_Task_PointerNU_Current_Task_Pointer());
    }

    /* call network initialization */

    init_struct.net_buffered_mem = &DEMOI_NET_Memory_Pool;
    init_struct.net_internal_mem = &System_Memory;

    if (NU_Init_Net(&init_struct) != NU_SUCCESS)
    {
        printf("Error at call to NU_Attach_IP_To_Device().\n");
        NU_SUSPEND_Task(NU_Current_Task_PointerNU_Current_Task_Pointer());
    }

    devices[0].dv_name = "NE2000_0";
    devices[0].dv_hw.ether.dv_irq = 5;
    devices[0].dv_hw.ether.dv_io_addr = 0x0300L;
    devices[0].dv_hw.ether.dv_shared_addr = 0;
    devices[0].dv_init = NE2000_Init;
    devices[0].dv_flags = 0;
    memcpy (devices[0].dv_ip_addr, "\0\0\0\0", 4);
    memcpy (devices[0].dv_subnet_mask, subnet, 4);

    /* Initialize the ethernet device. */
    devices[1].dv_name = "NE2000_1";
    devices[1].dv_hw.ether.dv_irq = 11;
    devices[1].dv_hw.ether.dv_io_addr = 0x0320L;
    devices[1].dv_hw.ether.dv_shared_addr = 0;
    devices[1].dv_init = NE2000_Init;

```

```
memcpy (devices[1].dv_ip_addr, "\0\0\0\0", 4);
memcpy (devices[1].dv_subnet_mask, subnet, 4);
devices[1].dv_flags = 0;

/* Initialize the hardware interfaces. */
NU_Init_Devices(devices, 2);

/* Allocate Memory for the NU_BOOTP_STRUCT. The memory for the
   structure is allocated on based on the number of devices that will
   have the Bootp Service performed on. */
status = NU_Allocate_Memory(&System_Memory, (VOID **)&bootp_ptr,
                           sizeof(NU_BOOTP_STRUCT) * DEVICE_COUNT,
                           NU_NO_SUSPEND);

if (status != NU_SUCCESS)
{
    printf("Cannot allocate memory for bootp structure.\n");
}

/* set all BOOTP fields to zero value */
memset(bootp_ptr, 0, sizeof(NU_BOOTP_STRUCT)* DEVICE_COUNT);

/* put in user callback function for the vendor options. */
/* bootp_ptr struct above is left blank unless the caller wanted to pass
   requested options to the BOOTP server. */

ret = NU_Bootp(devices[0].dv_name, bootp_ptr);

if (ret != NU_SUCCESS)
{
    printf("Bootp failed to resolve an IP address.\n");
}

/* Increment the NU_BOOTP_STRUCT to the next device. */
bootp_ptr++;

ret = NU_Bootp(devices[1].dv_name, bootp_ptr);

if (ret != NU_SUCCESS)
{
    printf("Bootp failed to resolve an IP address.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_TFTPGet2 \(IPv4/IPv6\)](#)

[NU\\_TFTPPut2 \(IPv4/IPv6\)](#)

## NU\_Clear\_Internal\_Log

This routine will clear all current errors from the NLOG\_Entry\_List array.

### Usage

```
STATUS NLOG_Clear_All_Errors (VOID);
```

### Example

```
STATUS status;  
status = NLOG_Clear_All_Errors();
```

### Related Topics

[Net API Functions](#)

## NU\_Close\_Socket (IPv4/IPv6)

This function is responsible for breaking the given socket connection on TCP or a socket allocation from UDP or Raw IP communication.

### Usage

```
STATUS NU_Close_Socket (INT socketd);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_NOT_CONNECTED`  
The socket is not connected.
  - `NU_SOCKET_CLOSED`  
The socket has been closed by another process.

### Description

The client and server must call the function to terminate a connection, whether a connection-oriented or connectionless transfer had been established. When using a TCP connection, the `NU_Close_Socket` service call will suspend until the connection is completely closed. In the case of UDP services, the socket and port entries are closed immediately.

### Example

```
INT socketd;    /* the socket descriptor */
int status;     /* return status of the NU_Close_Socket call */

status = NU_Close_Socket(socketd);

/* If status equals NU_SUCCESS, the NU_Close_Socket call was
   successful and the connection is terminated */
```

## Related Topics

[Net API Functions](#)

[NU\\_Shutdown \(IPv4/IPv6\)](#)

[NU\\_Abort \(IPv4/IPv6\)](#)

## NU\_CMSG\_DATA (IPv6)

This function returns a pointer to the data (called the `cmsg_data[]` member, even though such a member is not defined in the structure) following a [cmsg\\_hdr](#) structure.

### Usage

```
VOID *NU_CMSG_DATA (cmsg_hdr *cmsg);
```

### Arguments

- `cmsg`  
Pointer to the [cmsg\\_hdr](#) structure from which you want the data portion.

### Return Values

- Upon successful completion, this routine returns a pointer to the data portion of the data structure.

### Example

```
/* Use NU_CMSG_DATA to get the data portion when building a source address
ancillary data object */

msg_hdr    msg;          /* msg_hdr structure                */
cmsg_hdr    *cmsg;
CHAR        anc_buff; /* Buffer for ancillary data */
in6_pktinfo *pkt_info;

/* Allocate memory for the two pieces of ancillary data */
NU_Allocate_Memory(&System_Memory, (VOID**)&anc_buff,
                  NU_CMSG_SPACE(sizeof(in6_pktinfo)) +
                  NU_CMSG_SPACE(sizeof(INT)), NU_NO_SUSPEND);

/* Set up the msg_hdr to point to the buffer of ancillary data */
msg.msg_control = anc_buff;
msg.msg_controllen = sizeof(anc_buff);

/* Get a pointer to the first cmsg_hdr in the buffer */
cmsg = NU_CMSG_FIRSTHDR(&msg);

/* Fill in the source address */
pkt_info = (in6_pktinfo*)NU_CMSG_DATA(cmsg);

pkt_info->in6_addr.family = NU_FAMILY_IP6;
memcpy(pkt_info->in6_addr.id, source_address, IP6_ADDR_LEN);
```

## Related Topics

[Net API Functions](#)

[NU\\_CMSG\\_NXTHDR \(IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_FIRSTHDR \(IPv6\)](#)

[NU\\_CMSG\\_LEN \(IPv6\)](#)

[NU\\_CMSG\\_SPACE \(IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)



## NU\_CMSG\_FIRSTHDR (IPv6)

This function returns a pointer to the first [cmsghdr](#) structure in the [msghdr](#) structure pointed to by msg.

### Usage

```
cmsghdr *NU_CMSG_FIRSTHDR (msghdr *msg);
```

### Arguments

- msg  
Pointer to the [msghdr](#) data structure which contains a pointer to the ancillary data.

### Return Values

- A pointer to the first [cmsghdr](#) structure in the buffer of ancillary data.  
Upon successful completion.
- NU\_NULL  
If there is no data in the buffer.

### Description

The macro returns NULL if there is no ancillary data pointed to the by [msghdr](#) structure (that is, if either msg\_control is NULL or if msg\_controllen is less than the size of a [cmsghdr](#) structure).

### Example

```
/* Begin filling in the members of the ancillary data structure by getting
a pointer to the first cmsghdr structure */

msghdr  msg;          /* msghdr structure */
cmsghdr *cmsg;
CHAR    anc_buff;     /* Buffer for ancillary data */

/* Allocate memory for the two pieces of ancillary data */
NU_Allocate_Memory(&System_Memory, (VOID**)&anc_buff,
                  NU_CMSG_SPACE(sizeof(in6_pktinfo)) +
                  NU_CMSG_SPACE(sizeof(INT)), NU_NO_SUSPEND);

/* Set up the msghdr to point to the buffer of ancillary data */
msg.msg_control = anc_buff;
msg.msg_controllen = sizeof(anc_buff);

/* Get a pointer to the first cmsghdr in the buffer */
cmsg = NU_CMSG_FIRSTHDR(&msg);
```

## Related Topics

[Net API Functions](#)

[NU\\_CMSG\\_NXTHDR \(IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_CMSG\\_LEN \(IPv6\)](#)

[NU\\_CMSG\\_SPACE \(IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

## NU\_CMSG\_LEN (IPv6)

Given the length of an ancillary data object, this function returns the value to store in the `cmsg_len` member of the `cmsg_hdr` structure, taking into account any padding needed to satisfy alignment requirements.

### Usage

```
INT32 NU_CMSG_LEN (INT32 data_length);
```

### Arguments

- `data_length`  
The number of bytes of ancillary data that will be put in the `cmsg_hdr` structure.

### Return Values

- The number of bytes to store in the `cmsg_len` member of the `cmsg_hdr` structure.  
Upon successful completion.

### Example

```
/* Fill in the cmsg_len field of the cmsghdr structure for the source
address ancillary data structure */

msg_hdr    msg;          /* msg_hdr structure          */
cmsghdr    *cmsg;
CHAR       anc_buff; /* Buffer for ancillary data */

/* Build an IPv6 UDP socket.*/

/* Allocate memory for the two pieces of ancillary data */
NU_Allocate_Memory(&System_Memory, (VOID**) &anc_buff,
                  NU_CMSG_SPACE(sizeof(in6_pktinfo)) +
                  NU_CMSG_SPACE(sizeof(INT)), NU_NO_SUSPEND);

/* Set up the msg_hdr to point to the buffer of ancillary data */
msg.msg_control = anc_buffer;
msg.msg_controllen = sizeof(anc_buffer);

/* Get a pointer to the first cmsghdr in the buffer */
cmsg = NU_CMSG_FIRSTHDR(&msg);

/* Fill in the source address */
pkt_info = (in6_pktinfo*) NU_CMSG_DATA(cmsg);

pkt_info->in6_addr.family = NU_FAMILY_IP6;
memcpy(pkt_info->in6_addr.id, source_address, IP6_ADDR_LEN);

/* Set the length of the source address object */
cmsg->cmsg_len = NU_CMSG_LEN(sizeof(in6_pktinfo));
```

## Related Topics

[Net API Functions](#)

[NU\\_CMSG\\_NXTHDR \(IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_CMSG\\_FIRSTHDR \(IPv6\)](#)

[NU\\_CMSG\\_SPACE \(IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

## NU\_CMSG\_NXTHDR (IPv6)

This function returns a pointer to the [cmsghdr](#) structure describing the next ancillary data object.

### Usage

```
cmsghdr *NU_CMSG_NXTHDR (msghdr *msg,  
                        cmsghdr *cmsg);
```

### Arguments

- `msg`  
Pointer to the [msghdr](#) data structure which contains a pointer to the ancillary data.
- `cmsg`  
Pointer to the current [cmsghdr](#) structure which you want next.

### Return Values

- A pointer to the next [cmsghdr](#) structure in the buffer of ancillary data.  
Upon successful completion.
- `NU_NULL`  
If there is no next structure in the buffer.

### Example

```
/* Get the next cmsghdr when building an ancillary data object */  
  
msghdr    msg;          /* msghdr structure          */  
cmsghdr   *cmsg;  
CHAR      anc_buff; /* Buffer for ancillary data */  
  
/* Allocate memory for the two pieces of ancillary data */  
NU_Allocate_Memory(&System_Memory, (VOID**)&anc_buff,  
                  NU_CMSG_SPACE(sizeof(in6_pktinfo)) +  
                  NU_CMSG_SPACE(sizeof(INT)), NU_NO_SUSPEND);  
  
/* Set up the msghdr to point to the buffer of ancillary data */  
msg.msg_control = anc_buff;  
msg.msg_controllen = sizeof(anc_buff);  
  
/* Get a pointer to the first cmsghdr in the buffer */  
cmsg = NU_CMSG_FIRSTHDR(&msg);  
  
/* Fill in the buffer of ancillary data */  
  
/* Get a pointer to the next cmsghdr structure */  
cmsg = CMSG_NXTHDR(&msg, cmsg);
```

## Related Topics

[Net API Functions](#)

[NU\\_CMSG\\_LEN \(IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_CMSG\\_FIRSTHDR \(IPv6\)](#)

[NU\\_CMSG\\_SPACE \(IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

## NU\_CMSG\_SPACE (IPv6)

Given the length of an ancillary data object, this function returns the space required by the object and its [cmsghdr](#) structure, including any padding needed to satisfy alignment requirements.

### Usage

```
INT32 NU_CMSG_SPACE (INT32 data_length);
```

### Arguments

- `data_length`  
The number of bytes of ancillary data that will be put in the [cmsghdr](#) structure.

### Return Values

- The number of bytes required for the [cmsghdr](#) structure.  
Upon successful completion.

### Example

```
/* Allocate a buffer large enough for all ancillary data by using the
macro NU_CMSG_SPACE for each piece of ancillary data */

msghdr   msg;          /* msghdr structure           */
cmsghdr  *cmsg;
CHAR     anc_buff;     /* Buffer for ancillary data */

/* Allocate memory for the two pieces of ancillary data */
NU_Allocate_Memory(&System_Memory, (VOID**)&anc_buff,
                  NU_CMSG_SPACE(sizeof(in6_pktinfo)) +
                  NU_CMSG_SPACE(sizeof(INT)), NU_NO_SUSPEND);

/* Set up the msghdr to point to the buffer of ancillary data */
msg.msg_control = anc_buffer;
msg.msg_controllen = sizeof(anc_buffer);
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_CMSG_FIRSTHDR (IPv6)</a>
<a href="#">NU_CMSG_LEN (IPv6)</a>	<a href="#">NU_CMSG_NXTHDR (IPv6)</a>
<a href="#">NU_Recvmsg (IPv4/IPv6)</a>	<a href="#">NU_Sendmsg (IPv4/IPv6)</a>
<a href="#">NU_CMSG_DATA (IPv6)</a>	

## NU\_Configure\_IPv6\_Interface (IPv6)

This function configures the IPv6-specific features of an interface.

### Usage

```
STATUS NU_Configure_IPv6_Interface (UINT32 device_index,  
                                   INT      opt_name,  
                                   VOID     *opt_val);
```

### Arguments

- **device\_index**  
The index of the interface to configure.
- **opt\_name**  
The feature to configure. The only current valid value for this variable is `IP6_AUTO_ADDR_CONFIG`, which is used to configure the creation of global and site-local addresses during Stateless Address Autoconfiguration. This option is enabled by default and can be disabled through this API or at the call to [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) by setting the `DV6_DISABLE_ADDR_CONFIG` flag in the `dv6_flags` member of the [NU\\_DEVICE](#) data structure. The [NU\\_DEVICE](#) data structure is described in the [Net Data Structures](#) section.
- **opt\_val**  
A pointer to the configuration value. A value of 1 will enable the feature on the interface. A value of 0 will disable the feature on the interface.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, the interface will be configured according to the user input parameters. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
Either the value specified in `opt_name` is not supported or `device_index` does not reference a valid interface in the system.

### Example

```
INT      opt_val;  
STATUS   status;  
UINT32   dev_index = 1;  
  
opt_val = 1;  
  
/* Configure the interface to enable the creation of global  
 * and site-local IPv6 addresses during Stateless Address  
 * Autoconfiguration.  
 */  
status = NU_Configure_IPv6_Interface(dev_index, IP6_AUTO_ADDR_CONFIG,
```



```
                                &opt_val);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Configure_IPv6_Interface.\n");
}
```

## Related Topics

[Net API Functions](#)

## NU\_Configure\_Policy\_Table (IPv6)

This function configures the Policy Table used for source and destination address selection within IPv6.

### Usage

```
STATUS NU_Configure_Policy_Table (IP6_POLICY_ENTRY *plcy_ptr,  
                                UINT32             action);
```

### Arguments

- plcy\_ptr  
A pointer to the policy to be configured. The [IP6\\_POLICY\\_ENTRY](#) data structure is defined in the [Net Data Structures](#) section.
- action

The action to perform on the policy specified by plcy\_ptr. Valid values are:

IP6\_ADD\_POLICY – Add the policy to the table

IP6\_DELETE\_POLICY – Delete the policy from the table

IP6\_UPDATE\_POLICY – Update the existing policy according to the values set in plcy\_ptr.

### Return Values

- NU\_SUCCESS  
Upon successful completion, and the Policy Table is updated. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Description

The default Policy Table is configured in the *networking/netv6/src/dflt\_plcy\_tbl.c* file according to the default parameters specified by RFC 3484:

**Table 2-28. Default Policy Table Parameters**

Prefix	Precedence	Label
::1/128	50	0
::/0	40	1
2002::/16	30	2
::/96	20	3
::ffff:0:0/96	10	4

The Policy Table is used to influence source address selection within the Nucleus IPv6 stack and destination address sorting for DNS look-ups.

The Policy Table is a longest-matching-prefix lookup table. Given an address A, a lookup in the policy table produces two values; a precedence value and a label. The precedence value is used for sorting destination addresses. If the precedence value of address A is greater than the precedence value of address B, address A is preferred over address B. The label value allows for policies that prefer a particular source address prefix for use with a destination address prefix. The algorithm prefers to use a source address S with a destination address D if the label value of S is equal to the label value of D.

### Example

```
STATUS          status;
IP6_POLICY_ENTRY plcy_ptr;
UINT8           new_prefix[] =
    {0x3f, 0xfe, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

/* Copy the prefix into the policy data structure. */
memcpy(plcy_ptr.prefix, new_prefix, IP6_ADDR_LEN);

/* Set the prefix length of the policy entry. */
plcy_ptr.prefix_len = 64;

/* Set the precedence of the policy entry. */
plcy_ptr.precedence = 30;

/* Set the label of the policy entry. */
plcy_ptr.label = 3;

/* Add the new policy to the table. */
status = NU_Configure_Policy_Table(&plcy_ptr, IP6_ADD_POLICY);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Configure_Policy_Table.\n");
}

/* Change the label of the policy. */
plcy_ptr.label = 10;

/* Update the policy with the new label value. */
status = NU_Configure_Policy_Table(&plcy_ptr, IP6_UPDATE_POLICY);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Configure_Policy_Table.\n");
}

/* Delete the policy from the table. */
status = NU_Configure_Policy_Table(&plcy_ptr, IP6_DELETE_POLICY);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Configure_Policy_Table.\n");
}
```

## Related Topics

[Net API Functions](#)

## NU\_Configure\_Router (IPv6)

This function configures router-specific parameters for an IPv6-enabled interface.

### Usage

```
STATUS NU_Configure_Router (UINT32      dev_index,
                           UINT32      flags,
                           INT         flag_opts,
                           DEV6_RTR_OPTS *rtr_opts);
```

### Arguments

- **dev\_index**

The index of the device to configure.

- **flags**

Flags to set or remove from the interface:

**DV6\_ISROUTER**

Transmit Router Advertisement messages and process Router Solicitation messages on this interface.

**DV6\_ADV\_MGD**

Set the Managed Address Configuration flag in outgoing Router Advertisement messages.

**DV6\_ADV\_OTH\_CFG**

Set the Other Stateful Configuration flag in outgoing Router Advertisement messages.

- **flag\_opts**

Instructions on what to do with the flags specified:

**DV6\_REM\_FLGS**

Remove the indicated flags from the interface.

**DV6\_ADD\_FLGS**

Add the indicated flags to the interface.

**DV6\_IGN\_FLGS**

Leave the flags on the interface unmodified.

- **rtr\_opts**

A pointer to the parameters to configure. The [DEV6\\_RTR\\_OPTS](#) data structure is defined in the [Net Data Structures](#) section.

## Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- NU\_INVALID\_PARM

The device is invalid, rtr\_opts is NULL, or the length of a specified prefix is invalid.

- NU\_NO\_MEMORY

Insufficient memory to add the prefixes specified.

## Example

```
/* Enable router advertisements */

UINT32      flags, dev_index;
DEV6_RTR_OPTS  rtr_opts;

/* Initialize the device */
. . .

/* Get the Nucleus NET assigned index value of the interface from
 * the name of the interface.
 */
dev_index = NU_IF_NameToIndex("if_0");

/* Set up the parameters for the router */
flags = (DV6_ISROUTER | DV6_ADV_MGD | DV6_ADV_OTH_CFG);
rtr_opts.rtr_MaxRtrAdvInterval = 30;
rtr_opts.rtr_MinRtrAdvInterval = 3;
rtr_opts.rtr_AdvLinkMTU = 0;          /* Do not send Link MTU option */
rtr_opts.rtr_AdvReachableTime = 3000;
rtr_opts.rtr_AdvRetransTimer = 18000;
rtr_opts.rtr_AdvCurHopLimit = -1;    /* Use default value */
rtr_opts.rtr_AdvDefaultLifetime = 300000;

/* Configure the router to transmit Router Advertisement
 * messages with the above parameters.
 */
NU_Configure_Router(dev_index, flags, DV6_ADD_FLGS, &rtr_opts);
```

## Example

```
/*Disable Router Advertisements */

UINT32      dev_index;
DEV6_RTR_OPTS  rtr_opts;

/* Initialize the device */
. . .

/* Get the Nucleus NET assigned index value of the interface from
 * the name of the interface.
 */
```

```
dev_index = NU_IF_NameToIndex("if_0");

/* Do not modify any of the configured parameters */
rtr_opts.rtr_MaxRtrAdvInterval = -2;
rtr_opts.rtr_MinRtrAdvInterval = -2;
rtr_opts.rtr_AdvLinkMTU = -2;
rtr_opts.rtr_AdvReachableTime = -2;
rtr_opts.rtr_AdvRetransTimer = -2;
rtr_opts.rtr_AdvCurHopLimit = -2;
rtr_opts.rtr_AdvDefaultLifetime = -2;

/* Configure the router to stop transmitting Router
 * Advertisement messages.
 */
NU_Configure_Router(dev_index, DV6_ISROUTER, DV6_REM_FLGS, &rtr_opts);
```

## Related Topics

[Net API Functions](#)

[NU\\_Add\\_Prefix\\_Entry \(IPv6\)](#)

## NU\_Connect (IPv4/IPv6)

This function is responsible for establishing a connection request.

### Usage

```
STATUS NU_Connect (INT          socketd,  
                  struct addr_struct *servaddr,  
                  INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **servaddr**  
Pointer to the server's protocol-specific address and family type in [addr\\_struct](#). You must fill in the family parameter with the family type of the foreign side of the connection; either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6, the port parameter with the port number of the foreign side of the connection, and the id.is\_ip\_addrs parameter with the IP address of the foreign side of the connection.
- **addrlen**  
This parameter is reserved for future use. A value of zero should be used.

### Return Values

- A socket descriptor with a value greater than or equal to 0.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
The server address parameter was not valid.
  - **NU\_NOT\_CONNECTED**  
The connection attempt failed. The socket being used for the connection must be closed by the application layer. Future connection attempts must be made with a new socket.
  - **NU\_INVALID\_ADDRESS**  
The address does not match a class A, class B or class C address.
  - **-1**  
If the MEM\_Dequeue failed in TCPSS\_Send\_SYN\_FIN.



- **NU\_IS\_CONNECTING**  
The socket is non-blocking and the connection is being established.
- **NU\_SOCKET\_CLOSED**  
The socket has been closed by another process.
- **NU\_DEST\_UNREACH\_ADMIN**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_OPTION**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_NET**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_HOST**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PROT**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_FRAG**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_SRCFAIL**

An ICMP Error Code was received on the socket.

- NU\_PARM\_PROB

An ICMP Error Code was received on the socket.

- NU\_SOURCE\_QUENCH

An ICMP Error Code was received on the socket.

## Description

When calling NU\_Connect using a TCP socket, only the client must call the NU\_Connect() routine. When a connection-oriented transfer is being established over a TCP socket, the client and server may exchange messages agreeing on such items as buffer size and amount of data to be transferred. The only valid data transfer services that can be used with NU\_Connect over a TCP socket are [NU\\_Send \(IPv4/IPv6\)](#), [NU\\_Sendmsg \(IPv4/IPv6\)](#), [NU\\_Recv \(IPv4/IPv6\)](#), and [NU\\_Recvmsg \(IPv4/IPv6\)](#).

When calling [NU\\_Connect \(IPv4/IPv6\)](#) using a UDP socket, the client and/or the server can call the [NU\\_Connect \(IPv4/IPv6\)](#) routine to set the socket's peer address and port number. This designates the recipient of all datagrams sent on subsequent data transmissions and limits the remote sender when receiving data. If a source IP address is not specified via a call to [NU\\_Bind \(IPv4/IPv6\)](#), [NU\\_Connect \(IPv4/IPv6\)](#) will select a source IP address based on the route found to the peer address. If the address passed in is the wildcard address for the specified family, the socket's peer address is reset. Note that when using UDP sockets with [NU\\_Connect \(IPv4/IPv6\)](#), no actual physical connection is made with the remote peer. A connected UDP socket may use [NU\\_Send \(IPv4/IPv6\)](#), [NU\\_Send\\_To \(IPv4/IPv6\)](#), [NU\\_Sendmsg \(IPv4/IPv6\)](#), [NU\\_Recv \(IPv4/IPv6\)](#), [NU\\_Recv\\_From \(IPv4/IPv6\)](#), and [NU\\_Recvmsg \(IPv4/IPv6\)](#) to send and receive data. Note that when using [NU\\_Send \(IPv4/IPv6\)](#) and [NU\\_Recv \(IPv4/IPv6\)](#) with connected UDP sockets, the routines operate faster than [NU\\_Send\\_To \(IPv4/IPv6\)](#) and [NU\\_Recv\\_From \(IPv4/IPv6\)](#) since the peer and source information is already known.

## Example - IPv4

```
STATUS status;          /* return status of the NU_Connect function */
INT  socketd;           /* the socket descriptor */
struct addr_struct *servaddr; /* the server address structure */
.
.
.
/* the user must fill in the address of the server with which it
   is requesting a connection before making the NU_Connect call */

servaddr->family = NU_FAMILY_IP; /* Specifies IPv4 */
servaddr->port = 23; /* Telnet's port number */
servaddr->id.is_ip_addrs[0] = 192;
servaddr->id.is_ip_addrs[1] = 9; /* local machine's */
servaddr->id.is_ip_addrs[2] = 200; /* 4-digit IP number */
servaddr->id.is_ip_addrs[3] = 3;

status = NU_Connect(socketd, servaddr, 0);
```

```
/* if status is greater than or equal to 0, the NU_Connect call
   was successful and the client is connected to the specified
   server */
```

## Example - IPv6

```
STATUS status;          /* return status of the NU_Connect function */
INT  socketd;           /* the socket descriptor */
struct addr_struct *servaddr; /* the server address structure*/
.
.
.
/* the user must fill in the address of the server with which it
   is requesting a connection before making the NU_Connect call */

servaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 */
servaddr->port = 23; /* Telnet's port number */
servaddr->id.is_ip_addrs[0] = 0xfe;
servaddr->id.is_ip_addrs[1] = 0x80; /* local machine's */
servaddr->id.is_ip_addrs[2] = 0; /* 16-digit IP number */
servaddr->id.is_ip_addrs[3] = 0;
servaddr->id.is_ip_addrs[4] = 0;
servaddr->id.is_ip_addrs[5] = 0;
servaddr->id.is_ip_addrs[6] = 0;
servaddr->id.is_ip_addrs[7] = 0;
servaddr->id.is_ip_addrs[8] = 0;
servaddr->id.is_ip_addrs[9] = 0;
servaddr->id.is_ip_addrs[10] = 0xa;
servaddr->id.is_ip_addrs[11] = 0xb;
servaddr->id.is_ip_addrs[12] = 0xff;
servaddr->id.is_ip_addrs[13] = 0x2;
servaddr->id.is_ip_addrs[14] = 0xc;
servaddr->id.is_ip_addrs[15] = 0xd;
status = NU_Connect(socketd, servaddr, 0);

/* if status is greater than or equal to 0, the NU_Connect call
   was successful and the client is connected to the specified
   server */
```

## Related Topics

[Net API Functions](#)

[NU\\_Bind \(IPv4/IPv6\)](#)

[NU\\_Listen \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

[NU\\_Accept \(IPv4/IPv6\)](#)

## NU\_Create\_IPv4\_Mapped\_Addr (IPv6)

This function will create an IPv4-Mapped IPv6 address from the IPv4 address provided.

### Usage

```
VOID NU_Create_IPv4_Mapped_Addr (UINT8 *ipv6_addr,  
                                UINT32 ipv4_addr);
```

### Arguments

- `ipv6_addr`  
A pointer to the memory in which to create the IPv4-Mapped IPv6 address.
- `ipv4_addr`  
The IPv4 address from which to create the IPv4-Mapped IPv6 address.

### Example

```
UINT8    ipv4_addr[] = {192, 200, 100, 1};  
UINT8    ipv6_addr[IP6_ADDR_LEN];  
  
NU_Create_IPv4_Mapped_Addr(ipv6_addr, IP_ADDR(ipv4_addr));
```

### Related Topics

[Net API Functions](#)

[NU\\_Is\\_IPv4\\_Mapped\\_Addr \(IPv6\)](#)

## NU\_Delete\_DNS\_Server (IPv4)

This function will delete an IPv4 DNS server from the list of IPv4 DNS servers that will be used when trying to resolve host addresses or host names.

### Usage

```
STATUS NU_Delete_DNS_Server (UINT8 *dns_ip);
```

### Arguments

- `dns_ip`  
This is a pointer to the address of the DNS server to be deleted.

### Return Values

- `NU_SUCCESS`  
Upon successful completion, the DNS server will have been deleted from the list. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_PARM`  
The IP address was not valid. It was not found in the list of DNS servers.

### Example

```
UINT8      dns1[] = {192,200,100,1};
STATUS     status;

status = NU_Delete_DNS_Server(dns1);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Delete_DNS_Server().\n");
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Add_DNS_Server2 (IPv4/IPv6)</a>
<a href="#">NU_Delete_DNS_Server2 (IPv4/IPv6)</a>	<a href="#">NU_Free_Host_Entry (IPv4/IPv6)</a>
<a href="#">NU_Get_DNS_Servers (IPv4)</a>	<a href="#">NU_Get_DNS_Servers2 (IPv4/IPv6)</a>
<a href="#">NU_Add_DNS_Server (IPv4)</a>	

## NU\_Delete\_DNS\_Server2 (IPv4/IPv6)

This function is used for IPv4 and IPv6 and replaces the function NU\_Delete\_DNS\_Server. This function will delete a DNS server from the list of IPv4 or IPv6 DNS servers that will be used when trying to resolve host addresses or host names.

### Usage

```
STATUS NU_Delete_DNS_Server2 (UINT8 *dns_ip,  
                             INT16 family);
```

### Arguments

- dns\_ip  
This is a pointer to the address of the DNS server to be deleted.
- family  
The family type of the DNS Server to delete – either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6.

### Return Values

- NU\_SUCCESS  
Upon successful completion, the DNS server will have been deleted from the list specified. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - NU\_INVALID\_PARM  
The IP address was not valid. It was not found in the list of DNS servers.

### Example - IPv4

```
UINT8      dns1[] = {192,200,100,1};  
STATUS     status;  
  
/* Delete the IPv4 DNS Server from the list of DNS servers */  
status = NU_Delete_DNS_Server2(dns1, NU_FAMILY_IP);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Delete_DNS_Server2().\n");  
}
```

### Example - IPv6

```
UINT8      dns1[] = {0xfe,0x80,0,0,0,0,0,0,0,0,0,0,0xa,0xb,0xff,2,0xc,0xd};  
STATUS     status;  
  
/* Delete the IPv6 DNS Server from the list of DNS servers */  
status = NU_Delete_DNS_Server2(dns1, NU_FAMILY_IP6);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Delete_DNS_Server2().\n");  
}
```

}

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Get\\_DNS\\_Servers \(IPv4\)](#)

[NU\\_Add\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

[NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)

## NU\_Delete\_Host\_Entry (IPv4/IPv6)

This function deletes a DNS host entry from the DNS Host List.

### Usage

```
STATUS NU_Delete_Host_Entry (NU_HOSTENT *host_entry);
```

### Arguments

- `host_entry`

A pointer to the DNS host entry to delete. The structure [NU\\_HOSTENT](#) is defined in the [Net Data Structures](#) section.

### Return Values

- `NU_SUCCESS`

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- `NU_INVALID_PARM`

The input parameter `host_entry` is NULL or no matching DNS host entry was found in the DNS Host List.

### Example

```
NU_HOSTENT *host_entry;
STATUS      status;

/* Resolve a host by name. */
host_entry = NU_Get_IP_Node_By_Name("mentor.com", NU_FAMILY_IP, 0,
&status);

/* If the host was successfully resolved. */
if (host_entry)
{
    . . .

    /* Delete the host entry so it is removed from the DNS Host List. */
    status = NU_Delete_Host_Entry(host_entry);

    if (status != NU_SUCCESS)
    {
        printf("Error at call to NU_Delete_Host_Entry.\n");
    }

    /* Free the memory that was allocated for the application when this
     * host was resolved by the DNS module.
     */
    NU_Free_Host_Entry(host_entry);
}
```



## Related Topics

[Net API Functions](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

## NU\_Delete\_Prefix\_Entry (IPv6)

This function deletes a prefix entry from the list of prefixes for the specified interface.

### Usage

```
STATUS NU_Delete_Prefix_Entry (UINT32 device_index,  
                               UINT8  *prefix);
```

### Arguments

- `device_index`  
The index of the device from which to delete the prefix entry.
- `prefix`  
A pointer to the prefix to delete.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - `NU_INVALID_PARM`  
One of the input parameters is invalid.

### Example

```
UINT8  prefix[] = {0x3f, 0xfe, 0, 0, 1, 2, 0, 0};  
UINT32 device_index;  
  
/* Initialize the device and flag it as a Router */  
. . .  
  
/* Get the Nucleus NET assigned interface index for the interface  
 * from the name assigned by the application.  
 */  
device_index = NU_IF_NameToIndex("if_0");  
  
/* Add the prefix */  
. . .  
  
/* Delete the prefix */  
NU_Delete_Prefix_Entry(device_index, prefix);
```

### Related Topics

[Net API Functions](#)

[NU\\_Add\\_Prefix\\_Entry \(IPv6\)](#)

## NU\_Delete\_Route (IPv4)

This function is responsible for deleting an IPv4 route that was previously added with the NU\_Add\_Route service.

### Usage

```
STATUS NU_Delete_Route (UINT8 *ip_dest);
```

### Arguments

- `ip_dest`  
Specifies the destination IP address for the route to delete. This should be the same address as the one used as the first parameter to the NU\_Add\_Route service.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_ADDRESS`  
A route to `ip_dest` could not be found in the routing table.
  - `NU_INVALID_PARM`  
`ip_dest` is a null pointer.

### Example

```
UINT8 router[] = {200,100,100,1};
UINT8 subnet_mask[] = {255,255,255,0};
UINT8 netwrk[] = {192,200,100,0};

/* Add a network route. */
NU_Add_Route(netwrk, subnet_mask, router);

. . .

/* Delete the network route */
NU_Delete_Route(netwrk);
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

## NU\_Delete\_Route2 (IPv4/IPv6)

This function is used for IPv4 and IPv6 and replaces the function NU\_Delete\_Route. This function is responsible for deleting a route that was previously added with the NU\_Add\_Route or NU\_Add\_Route6 service.

### Usage

```
STATUS NU_Delete_Route2 (UINT8 *ip_dest,  
                        UINT8 *gateway,  
                        INT16 family);
```

### Arguments

- **ip\_dest**  
Specifies the destination IP address for the route to delete.
- **gateway**  
Specifies the gateway address of the specific route associated with ip\_dest to delete. If this value is NU\_NULL, all routes associated with ip\_dest will be deleted.
- **family**  
The family type of the route to delete; either NU\_FAMILY\_IP to delete an IPv4 route or NU\_FAMILY\_IP6 to delete an IPv6 route.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_NOT\_FOUND**  
The route does not exist.
  - **NU\_INVALID\_PARM**  
ip\_dest is NULL or the family specified is invalid.

### Example - IPv4

```
UINT8 router[] = {200,100,100,1};  
UINT8 subnet_mask[] = {255,255,255,0};  
UINT8 netwrk[] = {192,200,100,0};  
  
/* Add a network route. */  
NU_Add_Route(netwrk, subnet_mask, router);  
  
. . .  
  
/* Delete the route just added */  
NU_Delete_Route2(netwrk, router, NU_FAMILY_IP);
```

## Example - IPv6

```
UINT8 dest[] =
    {0x3f,0xfe,0,1,0,2,0,0,0,0,0x12,0x34,0xff,0x56,0x67,0x90};
UINT8 gateway[] =
    {0xfe,0x80,0,0,0,0,0,0,0,0,0xa,0xb,0xff,0x2,0xc,0xd};

/* Add a host route. */
NU_Add_Route6(dest, gateway, 128);

. . .

/* Delete the route just added */
NU_Delete_Route2(dest, gateway, NU_FAMILY_IP6);
```

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

## NU\_Detach\_IP\_From\_Device (IPv4/IPv6)

This function disables a device, removes the attached non-DHCP IP address(es) from the device, and deletes all routes using the device.

### Usage

```
STATUS NU_Detach_IP_From_Device (CHAR *name);
```

### Arguments

- name  
A pointer to the name of the device.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
There is no device in the system with the associated name.

### Description

Note that the device still exists in the system, it is just not accepting or transmitting packets. No resources for the device are released. If the device contains an IP address obtained via DHCP, the application must use NU\_Dhcp\_Release() to delete the IP address from the device.

### Example

```
/* Disable the device */
if (NU_Detach_IP_From_Device ("Ethernet_0") != NU_SUCCESS)
{
    /* log an error */
    printf("Error at call to NU_Detach_IP_From_Device().\n");
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Attach_IP_To_Device (IPv4)</a>
<a href="#">NU_Device_Up (IPv4/IPv6)</a>	<a href="#">NU_Remove_Device (IPv4/IPv6)</a>
<a href="#">NU_Remove_IP_From_Device (IPv4/IPv6)</a>	<a href="#">NU_Add_IP_To_Device (IPv6)</a>

## NU\_Device\_Up (IPv4/IPv6)

This function checks the status of a device. A device is considered to be UP if it is in an operational state. Since PPP devices and DHCP-enabled devices can go down, this function can be used to check if that has occurred.

### Usage

```
STATUS NU_Device_Up (const CHAR *if_name);
```

### Arguments

- `if_name`  
Pointer to the name of the device to check.

### Return Values

- `NU_TRUE`  
The device is UP and usable.
- `NU_FALSE`  
The device has gone down for some reason. DHCP lease expired, PPP connection lost, and so on.
- `NU_INVALID_PARM`  
The `if_name` pointer is NULL or the name is unknown.

### Example

```
/* check the status of a device. */
if (NU_Device_Up("Ethernet_0") == NU_FALSE)
{
    /* log an error */
    printf("Ethernet_0 interface is not UP.\n");
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Attach_IP_To_Device (IPv4)</a>
<a href="#">NU_Init_Devices (IPv4/IPv6)</a>	<a href="#">NU_Remove_Device (IPv4/IPv6)</a>
<a href="#">NU_Remove_IP_From_Device (IPv4/IPv6)</a>	<a href="#">NU_Add_IP_To_Device (IPv6)</a>

## NU\_Dhcp (IPv4)

This service is used to dynamically configure the IPv4 address of an embedded device and/or retrieve other information from a DHCP server, such as a subnet mask, default gateway, and so on.

### Usage

```
STATUS NU_Dhcp (NU_DHCP_STRUCT *ds_ptr,
                CHAR            *dv_name);
```

### Arguments

- **ds\_ptr**  
This service is used to dynamically configure the IP address of an embedded device. It can also be used to retrieve other information from a DHCP server, such as a subnet mask, default gateway, and so on. [NU\\_DHCP\\_STRUCT](#) is defined in the [Net Data Structures](#) section.
- **dv\_name**  
A pointer to the name of the device on which to perform the DHCP query.  
The following members of the [NU\\_DHCP\\_STRUCT](#) must be filled in by the application:

**Table 2-29. NU\_DHCP\_STRUCT Application Layer Members**

Member	Description
*dhcp_opts	A pointer to the options to be added to the Discover Packet. This is filled in by the Application Layer. Refer to RFC 2132 for the available options and the syntax for using the respective options.
dhcp_opts_len	Length, in octets, of the dhcp_opts field. This is filled in by the Application Layer.

layer:

The following members are filled in by the stack:

**Table 2-30. NU\_DHCP\_STRUCT Stack Members**

Member	Description
dhcp_siaddr	The IP address of the DHCP Server which was queried.
dhcp_giaddr	The gateway IP address through which a default route was added to the Routing Table.
dhcp_yiaddr	The IP address assigned to the device by the DHCP Server.
dhcp_net_mask	The subnet mask of the IP address assigned by the DHCP Server.



Table 2-30. NU\_DHCP\_STRUCT Stack Members (cont.)

Member	Description
dhcp_xid	The random transaction ID generated by the stack to use for the most recent DHCP Request issued by this device.
dhcp_mac_addr	The MAC (link-layer) address of the device.
dhcp_sname	The host name of the DHCP Server.
dhcp_file	Fully pathed file name returned by the DHCP Server.
dhcp_secs	Seconds since boot began.

## Return Values

- NU\_SUCCESS

Upon successful completion, the local host's IP address is initialized and the [NU\\_DHCP\\_STRUCT](#) is completely filled in. The [NU\\_DHCP\\_STRUCT](#) can be used by the application to access all DHCP information. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- NU\_NO\_SOCKET\_MEMORY

There was not enough memory available to allocate a new socket descriptor structure.

- NU\_INVALID\_PROTOCOL

The combination of family and type parameters for the new socket mapped to a protocol, which was not supported.

- NU\_NO\_MEMORY

Allocation of memory failed.

- NU\_INVALID\_PARM

*ds\_ptr* or *dv\_name* are NULL.

- NU\_DHCP\_INIT\_FAILED

A resource (memory, socket, and so on) was not available or could not be initialized properly.

- NU\_DHCP\_REQUEST\_FAILED

A DHCP request was sent but no response was received.

## Description

The application layer must set the `dev_ip_addr` member of the [NU\\_DEVICE](#) structure to all zero's before the call to `NU_Init_Devices` for `NU_Dhcp` to request an IPv4 address from the DHCP Server.

Note that although Nucleus NET interfaces can be assigned multiple addresses, only one IP address can be assigned to the interface using the `NU_Dhcp()` routine.

## Example

```
#define DHCP_DEV_COUNT    2
NU_DHCP_STRUCT    *dhcp_ptr, *dsp;
NU_DEVICE          devices[DEVICE_COUNT];

/* These are the dhcp options desired. Each option is specified by three
   items, an option id, an option length, and an option value. The
   following specifies two options. The first option is to request that
   the DHCP server return some specific parameters.
   There are three such parameters (the length). The three requested are
   DHCP_NETMASK, DHCP_ROUTE, and DHCP_DNS. The second option,
   DHCP_HOSTNAME, is to specify a name that will be sent to the server.
   The name has 11 characters including the null terminator. The value is
   "DHCPCLIENT" + null terminator. */

UINT8 dhcp_options[] = {DHCP_REQUEST_LIST,3,DHCP_NETMASK, DHCP_ROUTE,
                        DHCP_DNS, DHCP_HOSTNAME,11,
                        'D','H','C',' ','P','C','L','I','E','N','T',0}

devices[0].dv_name = "NE2000_0";
devices[0].dv_hw.ether.dv_irq = 5;
devices[0].dv_hw.ether.dv_io_addr = 0x0300L;
devices[0].dv_hw.ether.dv_shared_addr = 0;
devices[0].dv_init = NE2000_Init;
devices[0].dv_flags = 0;
/* Must be set to all zero's to request an IP address */
memcpy(devices[0].dv_ip_addr, "\0\0\0\0", 4);
memcpy (devices[0].dv_subnet_mask, subnet, 4);

devices[1].dv_name = "NE2000_1";
devices[1].dv_hw.ether.dv_irq = 11;
devices[1].dv_hw.ether.dv_io_addr = 0x0320L;
devices[1].dv_hw.ether.dv_shared_addr = 0;
devices[1].dv_init = NE2000_Init;
devices[1].dv_flags = 0;
memcpy(devices[1].dv_ip_addr, "\0\0\0\0", 4);
memcpy (devices[1].dv_subnet_mask, subnet, 4);

/* Initialize the devices. */
NU_Init_Devices(devices, DHCP_DEV_COUNT);

/* allocate memory for the DHCP structures. */
status = NU_Allocate_Memory(&System_Memory, (VOID **)&dhcp_ptr,
                           sizeof(NU_DHCP_STRUCT)
                           * DHCP_DEV_COUNT, NU_NO_SUSPEND);

/* set all DHCP fields to zero value */
memset(dhcp_ptr, 0, sizeof(NU_DHCP_STRUCT) * DHCP_DEV_COUNT );
```

```
/* put in user callback function for the vendor options. */
for( i = 0, dsp = dhcp_ptr; i < DEVICE_COUNT; i++, dsp++ )
{
    /* Specify the DHCP options desired. */
    dsp->dhcp_opts = dhcp_options;
    dsp->dhcp_opts_len = sizeof(dhcp_options);
}

/* dhcp_ptr struct above is left blank unless the caller wanted to pass
requests to the DHCP server. Get the IP address and information from
the DHCP Server for the first device. */
do
{
    NU_Dhcp(dhcp_ptr, devices[0].dv_name);
    if(ret != NU_SUCCESS)
        NU_Sleep(TICKS_PER_SECOND);
}while(ret!= NU_SUCCESS);

/* If NU_Dhcp returns NU_SUCCESS then the IP address has been initialized
and DHCP structure filled */
```

## Related Topics

[Net API Functions](#)

[NU\\_Dhcp\\_Release \(IPv4\)](#)

## NU\_Dhcp6 (IPv6)

This function invokes DHCPv6 client processing on the specified interface. You can manually invoke Stateful Address Autoconfiguration or obtain configuration information using this API as described.

### Usage

```
STATUS NU_Dhcp6 (DHCP6_CLIENT *dhcp_cli_ptr,  
                UNSIGNED      suspend);
```

### Arguments

- dhcp\_cli\_ptr

A pointer to the [DHCP6\\_CLIENT](#) configuration data structure. The [DHCP6\\_CLIENT](#) data structure is defined in the [Net Data Structures](#) section.

- suspend

This variable controls whether the function will suspend pending completion of the DHCPv6 request. Set this value to [NU\\_SUSPEND](#) to suspend on the DHCPv6 request, or [NU\\_NO\\_SUSPEND](#) to return immediately, allowing the DHCPv6 request to complete in the background.

### Return Values

- [NU\\_SUCCESS](#)

Upon successful completion, the DHCPv6 client request will be issued, and the interface will be configured according to the server parameters. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- [NU\\_INVALID\\_PARM](#)

One of the input parameters is invalid.

- [NU\\_DHCP\\_REQUEST\\_FAILED](#)

The request failed. Either a server could not be found, or the server could not complete the request.

### Example

```
/* Request an IPv6 Address and DNS Server List */  
  
UINT8      dhcp_buffer[256];  
UINT32     iaid;  
UINT16     dns_server = DHCP6_OPT_DNS_SERVERS;  
DHCP6_CLIENT ds_ptr;  
STATUS     status;  
  
/* Get the index associated with this interface name. */  
ds_ptr.dhcp6_dev_index = NU_IF_NameToIndex("eth0");  
  
/* Get the IA ID associated with this interface. */  
status = NU_Get_DHCP_IAID(ds_ptr.dhcp6_dev_index, &iaid);
```

```
if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Get_DHCP_IAID.\n");
}

/* Build an IA_NA option. */
ds_ptr.dhcp6_opt_length = DHCP6_Build_IA_NA_Option(dhcp_buffer,
                                                    iaid, 0, 0, NU_NULL, 0);

/* Build the DNS Recursive Server Option, which is included in an Option
 * Request Option.
 */
ds_ptr.dhcp6_opt_length +=
    DHCP6_Build_Option_Request_Option(&dhcp_buffer[ds_ptr.dhcp6_opt_length],
                                       1, &dns_server);

/* Set a pointer to the buffer that will hold the DHCPv6 options. */
ds_ptr.dhcp6_user_opts = dhcp_buffer;

/* Issue the DHCPv6 client request, suspending on completion of the
 * request.
 */
status = NU_Dhcp6(&ds_ptr, NU_SUSPEND);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Dhcp6.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>	<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>
<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## NU\_Dhcp\_Release (IPv4)

This function is responsible for releasing an IPv4 address obtained from a Dynamic Host Configuration Protocol Server. The parameters contain information that was originally provided by the DHCP Server.

### Usage

```
STATUS NU_Dhcp_Release (NU_DHCP_STRUCT *ds_ptr,  
                        CHAR             *dv_name);
```

### Arguments

- **ds\_ptr**  
A pointer to the structure that was originally filled in by the call to [NU\\_Dhcp \(IPv4\)](#). The [NU\\_DHCP\\_STRUCT](#) is defined in the [Net Data Structures](#) section.
- **dv\_name**  
The name of the device for which the IP address is to be released.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
ds\_ptr or dv\_name are NULL.
  - **NU\_DHCP\_INIT\_FAILED**  
A resource (memory, socket, and so on) was not available or could not be initialized properly.
  - **NU\_NO\_BUFFERS**  
All of the NET buffers have been used, the release request could not be completed.

### Example

```
#define DHCP_DEV_COUNT          2  
  
NU_DHCP_STRUCT    *dhcp_ptr, *dsp;  
NU_DEVICE          devices[DEVICE_COUNT];  
  
devices[0].dv_name = "NE2000_0";  
devices[0].dv_hw.ether.dv_irq = 5;  
devices[0].dv_hw.ether.dv_io_addr = 0x0300L;  
devices[0].dv_hw.ether.dv_shared_addr = 0;  
devices[0].dv_init = NE2000_Init;  
devices[0].dv_flags = 0;  
memcpy(devices[0].dv_ip_addr, "\0\0\0\0", 4);  
memcpy (devices[0].dv_subnet_mask, subnet, 4);
```

```
devices[1].dv_name = "NE2000_1";
devices[1].dv_hw.ether.dv_irq = 11;
devices[1].dv_hw.ether.dv_io_addr = 0x0320L;
devices[1].dv_hw.ether.dv_shared_addr = 0;
devices[1].dv_init = NE2000_Init;
devices[1].dv_flags = 0;
memcpy(devices[1].dv_ip_addr, "\0\0\0\0", 4);
memcpy (devices[1].dv_subnet_mask, subnet, 4);

/* Initialize the devices. */
NU_Init_Devices(devices, DHCP_DEV_COUNT);

/* allocate memory for the DHCP structures. */
status = NU_Allocate_Memory(&System_Memory, (VOID **)&dhcp_ptr,

/* set all DHCP fields to zero value */
memset(dhcp_ptr, 0, sizeof(NU_DHCP_STRUCT) * DHCP_DEV_COUNT );

/* put in user callback function for the vendor options. */
for (i = 0, dsp = dhcp_ptr; i < DEVICE_COUNT; i++, dsp++)
{
    /* Specify the DHCP options desired. */
    dsp->dhcp_opts = dhcp_options;
    dsp->dhcp_opts_len = sizeof(dhcp_options);
}

/* dhcp_ptr struct above is left blank unless the caller wanted to
   pass requests to the DHCP server. Get the IP address and
   information from the DHCP for them first device. */
ret = NU_Dhcp(dhcp_ptr, devices[0].dv_name);

/* If NU_Dhcp returns NU_SUCCESS then the IP address has been
   initialized and the DHCP structure filled */

/* Release the device 0 DHCP obtained device. */
status = NU_Dhcp_Release(dhcp_ptr, devices[0].dv_name);

/* If NU_Dhcp_Release returns NU_SUCCESS then the IP address has
   been successfully released */
```

## Related Topics

[Net API Functions](#)

[NU\\_Dhcp \(IPv4\)](#)

## NU\_Dhcp6\_Shutdown (IPv6)

This function shuts down DHCPv6 client processing on an interface. All IPv6 addresses that were obtained via DHCPv6 will be deleted from the interface.

If NU\_Dhcp6\_Shutdown is called while active leases exist for IPv6 addresses attached to the device(s), a RELEASE packet is not sent to the DHCP server. This is because the release process will attempt to use already deallocated structures to process the release request. Be sure to call NU\_Remove\_IP\_From\_Device prior to calling NU\_Dhcp6\_Shutdown in order to send the release notice to the DHCP Server.

### Usage

```
STATUS NU_Dhcp6_Shutdown (UINT32 dev_index,  
                          UINT8  flags);
```

### Arguments

- dev\_index  
The index of the interface for which DHCPv6 client processing is to be shut down.
- flags  
This parameter is currently unused.

### Return Values

- NU\_SUCCESS  
Upon successful completion, the DHCPv6 client processing is shut down on the interface. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Example

```
DHCP6_CLIENT  ds_ptr;  
STATUS        status;  
.  
.  
.  
/* Shutdown DHCPv6 client processing on the interface. */  
  
DHCP6_CLIENT  ds_ptr;  
STATUS        status;  
  
.  
.  
.  
  
/* Shutdown DHCPv6 client processing on the interface, and transmit a  
 * Release message for each IPv6 address that was obtained via DHCPv6.  
 */  
status = NU_Dhcp6_Shutdown(ds_ptr.dhcp6_dev_index, 0);  
  
if (status != NU_SUCCESS)  
{
```



```
    printf("Error at call to NU_Dhcp6.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>	<a href="#">NU_Dhcp6 (IPv6)</a>
<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## NU\_DNS\_SD\_Browse

This routine invokes a thread that will browse for instances of a service on the local network.

### Usage

```
STATUS NU_DNS_SD_Browse(INT  action,  
                        CHAR  *type,  
                        CHAR  *domain);
```

### Arguments

- action

You can start and stop the service using the following options:

NU\_DNS\_START  
NU\_DNS\_STOP

- type

The name of the service for which to browse.

- domain

The domain in which to browse for the service. Only the domain “local” is currently supported.

### Return Values

- NU\_SUCCESS

The function completed successfully. The browse session has been started, and the calling thread will be signaled when a new record for the query is found.

- NU\_INVALID\_PARM

An input parameter is invalid.

- NU\_NO\_ACTION

The caller’s command to stop the service had no effect, because the system is not currently browsing for the service.

- Operating system specific error code, otherwise.

### Description

When a new instance of the service is found, or a change is identified in an already known service, a signal will be sent to the application thread that invoked this routine. Therefore, the caller must have registered a signal handler via `NU_Register_Signal_Handler()` and enabled the control signal `MDNS_SIGNAL` via `NU_Control_Signals()`.

---

#### Note



More information about `NU_Register_Signal_Handler()` and `NU_Control_Signals()` in the [Nucleus Kernel Guide](#).

---

## Example

```
STATUS status;

/* Set up the signal handler for the browse routine. */
NU_Register_Signal_Handler(Browse_Control);
NU_Control_Signals(1UL << MDNS_SIGNAL);

/* Browse for instances of the _ipp.tcp service.  When a matching record
 * is received, the routine Browse_Control will be notified
 */
status = NU_DNS_SD_Browse(NU_DNS_START, "_ipp._tcp", "local");
```

## Related Topics

[Net API Functions](#)

[NU\\_DNS\\_SD\\_Look\\_Up](#)

## NU\_DNS\_SD\_Look\_Up

This routine looks up the information necessary to contact and use the named service.

### Usage

```
NU_DNS_SD_SERVICE *NU_DNS_SD_Look_Up (CHAR    *name,  
                                       CHAR    *type,  
                                       CHAR    *domain,  
                                       STATUS  *status);
```

### Arguments

- name  
Pointer to the specific instance of the service as returned by [NU\\_DNS\\_SD\\_Refresh](#).
- type  
The service name.
- domain  
Pointer to the domain of the service. The domain “local” is the only domain supported at this time.
- status  
Pointer to the status of the call. Possible values upon return are:

NU\_SUCCESS

Upon successful completion.

NU\_INVALID\_PARM

If the input is invalid.

NU\_NO\_ACTION

If no records were found.

An operating system specific error code, otherwise.

### Return Values

- A pointer to the [NU\\_DNS\\_SD\\_SERVICE](#) structure filled in with the data necessary to connect to that service.

---

#### Note



The application must free the memory returned by this routine when finished with the data.

---

### Example

```
STATUS          status;  
NU_DNS_SD_INSTANCE *instance_ptr;  
NU_DNS_SD_SERVICE *service_ptr = NU_NULL;
```

```
/* Browse for instances of the _ipp.tcp service. When a matching
 * record is received, the routine Browse_Control will be notified and
 * will set Application_Sleep to NU_FALSE to wake up this thread.
 */
status = NU_DNS_SD_Browse(NU_DNS_START, "_ipp.tcp", "local");

while (Application_Sleep == NU_TRUE)
    NU_Sleep(TICKS_PER_SECOND);

/* Get a pointer to the instances of this record. */
instance_ptr = NU_DNS_SD_Refresh("_ipp.tcp", "local", &status);

/* Get the details of each record. */
while (instance_ptr)
{
    /* Truncate the type and domain. */
    instance_ptr->dns_sd_name[strlen(instance_ptr->dns_sd_name) -
                               strlen("_ipp.tcp.local")] = '\0';

    /* Look up the details associated with this instance of the
     * service.
     */
    service_ptr = NU_DNS_SD_Look_Up(instance_ptr->dns_sd_name,
                                     "_ipp.tcp", "local", &status);

    /* Do something with the service. */
    . . .

    /* The application must deallocate the memory returned from
     * NU_DNS_SD_Look_Up().
     */
    if (service_ptr)
    {
        NU_Deallocate_Memory(service_ptr);
    }

    instance_ptr = instance_ptr->dns_sd_next;
}

/* The application must deallocate the memory returned from
 * NU_DNS_SD_Refresh().
 */
if (instance_ptr)
{
    NU_Deallocate_Memory(instance_ptr);
}
```

## Related Topics

[Net API Functions](#)

[NU\\_DNS\\_SD\\_Refresh](#)

## NU\_DNS\_SD\_Refresh

This routine retrieves a list of instances on the network that offer the specified service.

### Usage

```
NU_DNS_SD_INSTANCE *NU_DNS_SD_Refresh (CHAR    *type,  
                                         CHAR    *domain,  
                                         STATUS  *status);
```

### Arguments

- type  
Pointer to the name of the service for which to retrieve the found instances.
- domain  
Pointer to the domain of scope for the respective instances. Only the domain “local” is currently supported.
- status  
Pointer to the status of the call. Possible values upon return are:

NU\_SUCCESS

Upon successful completion.

NU\_INVALID\_PARM

If the input is invalid.

NU\_NO\_ACTION

If no records were found.

An operating system specific error code, otherwise.

---

#### Note



Entries could still be returned in case of error, which indicates that memory could not be allocated for all available entries.

---

### Return values

- A list of instances that provide the service requested. The [NU\\_DNS\\_SD\\_INSTANCE](#) data structure is defined in the [Net Data Structures](#) section.
- NU\_NULL  
If no instances have been found.

---

#### Note



The application must free the memory returned by this routine when finished with the data.

---

## Description

The routine [NU\\_DNS\\_SD\\_Browse](#) must have been invoked prior to calling this routine, and the calling thread must have received a signal that records have been found corresponding to the prior [NU\\_DNS\\_SD\\_Browse](#) request.

The [NU\\_DNS\\_SD\\_INSTANCE](#) structure is defined in the [Net Data Structures](#) section.

## Example

```
STATUS          status;
NU_DNS_SD_INSTANCE *instance_ptr;

/* Browse for instances of the _ipp.tcp service.  When a matching
 * record is received, the routine Browse_Control will be notified and
 * will set Application_Sleep to NU_FALSE to wake up this thread.
 */
status = NU_DNS_SD_Browse(NU_DNS_START, "_ipp.tcp", "local");

while (Application_Sleep == NU_TRUE)
    NU_Sleep(TICKS_PER_SECOND);

/* Get a pointer to the instances of this record. */
instance_ptr = NU_DNS_SD_Refresh("_ipp.tcp", "local", &status);

/* Do something with the instances. */
. . .
```

## Related Topics

[Net API Functions](#)

[NU\\_DNS\\_SD\\_Look\\_Up](#)

## NU\_DNS\_Register\_Service

This routine registers and unregisters a service on the specified domain in the network using a specified network interface. When a service has been registered, the local node will answer DNS PTR queries for the indicated service. The same service can be registered across multiple interfaces.

### Usage

```
STATUS NU_DNS_SD_Register_Service(INT    action,  
                                  CHAR    *name,  
                                  CHAR    *type,  
                                  CHAR    *domain,  
                                  UINT16  port,  
                                  CHAR    *keys,  
                                  INT32   if_index);
```

### Arguments

- action

You can register and unregister the service using the following options:

NU\_DNS\_START  
NU\_DNS\_STOP

- name

The local identifier of the service instance. Any arbitrary string of length less than 64 bytes containing legal unicode characters.

- type

The name of the service to be registered.

- domain

The domain in which to register the service. Only the domain “local” is currently supported.

- port

The TCP or UDP port number upon which the service can be reached.

- keys

Additional (optional) key/value pairs to be stored in the TXT record of the advertised service. Each pair must be terminated with `\r\n`. The final string must be null-terminated.

- if\_index

Index of the interface on which the service will be advertised.

### Return Values

- NU\_SUCCESS

Function completed successfully. The service was registered or unregistered.



- **NU\_INVALID\_PARM**  
An input parameter is invalid.
- **NU\_NO\_ACTION**  
The service is already registered or unregistered.
- **NU\_NO\_MEMORY**  
There is not enough memory to register the service.
- Operating system specific error code, otherwise.

### Example

```
STATUS    status;
CHAR      key_buf[64];
INT32     key_size;

/* Build the key to pass into the routine. */
key_size = NU_DNS_Build_Key(key_buf,
                           "language=english\r\ncolor=blue\r\nfact=false\r\ncount=100",
                           64);
key_buf[key_size] = 0;

/* Register the service. */
status = NU_DNS_SD_Register_Service(NU_DNS_START, "this_service",
                                   "_ipp._tcp", "local", 1500,
                                   key_buf, 1);
```

### Related Topics

[Net API Functions](#)

[NU\\_DNS\\_Build\\_Key](#)

## NU\_DNS\_Build\_Key

This routine builds a buffer of key/value pairs per the format outlined in the DNS-SD RFC. The user may pass in n key/value pairs for formatting in a single call to the routine.

### Usage

```
INT32 NU_DNS_Build_Key(CHAR      *buffer,  
                      const char *kv_pair,  
                      INT32      length);
```

### Arguments

- **buffer**  
Pointer to the memory where the formatted keys will be stored.
- **kv\_pair**  
Pointer to a set of key/value pairs to be added to the buffer. They key/value pairs must be of the following format:  
  
"<key1>=<value1>\r\n<key2>=<value2>\r\n..."
- **length**  
Maximum size of the buffer.

### Return Values

- The size of the key that was added to the TXT record.
- **NU\_INVALID\_PARM**  
An input parameter is invalid.
- **NU\_NO\_MEMORY**  
The user buffer is too small.
- **NU\_INVALID\_SIZE**  
One of the key/value pairs is of invalid length.

### Example

```
CHAR  key_buf[64];  
INT32 key_size;  
  
key_size = NU_DNS_Build_Key(  
    key_buf,  
    "language=english\r\ncolor=blue\r\nfact=false\r\ncount=100",  
    64);
```

### Related Topics

[Net API Functions](#)

[NU\\_DNS\\_Parse\\_Key](#)

## NU\_DNS\_Parse\_Key

This routine parses out a single key/value pair from the buffer populated via [NU\\_DNS\\_SD\\_Look\\_Up](#).

### Usage

```
INT32 NU_DNS_Parse_Key (CHAR    *buffer,
                        INT32    offset,
                        CHAR    **out_key,
                        CHAR    **out_value);
```

### Arguments

- **buffer**  
A user buffer that gets populated by [NU\\_DNS\\_SD\\_Look\\_Up](#).
- **offset**  
The place in the buffer where the current key will be built.
- **out\_key**  
On return, a pointer to the key.
- **out\_value**  
On return, a pointer to the value of the key.

### Return Values

- The total size of the key/value data parsed so far.  
This return value can be used as an offset to fetch the next key/value pair from the same buffer.
- **NU\_INVALID\_PARM**  
An input parameter is invalid.

### Example

```
STATUS          status;
NU_DNS_SD_SERVICE *service_ptr;
NU_DNS_SD_INSTANCE *instance_ptr;
UINT8          instance_count;
CHAR           *key_ptr, *value_ptr;
INT32          offset;

/* Initiate a browsing session, and obtain a list of instances
 * on the network.
 */
. . .

/* Get the details of each record. */
while (instance_ptr)
{
    /* Truncate the type and domain. */
```

```
instance_ptr->dns_sd_name[strlen(instance_ptr->dns_sd_name) -
                             strlen("._ipp._tcp.local")] = '\0';

/* Look up the details associated with this instance of the
 * service.
 */
service_ptr = NU_DNS_SD_Look_Up(instance_ptr->dns_sd_name,
                                "_ipp._tcp", "local", &status);

if (service_ptr)
{
    /* If there are keys associated with this record. */
    if (service_ptr->dns_keys)
    {
        offset = 0;

        /* Parse through the keys. */
        do
        {
            offset = NU_DNS_Parse_Key(service_ptr->dns_keys, offset,
                                      &key_ptr, &value_ptr);

            if (offset < 0)
                break;
            else
            {
                /* Do something with the key. */
                . . .
            }
        } while (offset < service_ptr->dns_key_len);

        NU_Deallocate_Memory(service_ptr);
    }

    instance_ptr = instance_ptr->dns_sd_next;
}
```

## Related Topics

[Net API Functions](#)

[NU\\_DNS\\_Build\\_Key](#)

## NU\_Ethernet\_Link\_Down

This function brings the link down on the specified interface device.

### Usage

```
STATUS NU_Ethernet_Link_Down (CHAR *dev_name);
```

### Arguments

- `dev_name`  
This is a pointer to the name of the Ethernet device.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.

### Example

```
/* Initialize an interface. */  
. . .  
  
/* Disable the interface. */  
if (NU_Ethernet_Link_Down ("eth0") == NU_SUCCESS)  
{  
    . . .  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ethernet\\_Link\\_Up](#)

## NU\_Ethernet\_Link\_Up

This function brings the link up on the specified interface device.

### Usage

```
STATUS NU_Ethernet_Link_Up (CHAR *dev_name);
```

### Arguments

- dev\_name  
This is a pointer to the name of the Ethernet device.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- -1  
Otherwise.

### Example

```
/* Initialize an interface. */  
. . .  
  
/* Disable the interface. */  
. . .  
  
/* Re-enable the interface. */  
if (NU_Ethernet_Link_Up ("eth0") == NU_SUCCESS)  
{  
    . . .  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ethernet\\_Link\\_Down](#)

## NU\_Fcntl (IPv4/IPv6)

This function is responsible for performing miscellaneous actions on a socket. It may be called by both the client and the server. It is often desirable to determine if data is available from the network without blocking execution of the requesting task. For such cases, Nucleus NET provides a mechanism to specify whether or not reads are to be blocked. Using the NU\_Fcntl service call, you can specify if subsequent reads on the indicated socket are to be blocked.

### Usage

```
STATUS NU_Fcntl (INT    socketd,  
                INT16  command,  
                INT16  argument);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **command**  
Specifies the command requested by the application. Valid commands include: NU\_SETFLAG and NU\_SET\_ZC\_MODE.
- **argument**  
Specifies the command option. Valid arguments include: NU\_BLOCK, NU\_NO\_BLOCK, NU\_ZC\_ENABLE, and NU\_ZC\_DISABLE.

[Table 2-31](#) provides a list of commands and the associated arguments currently supported by NU\_Fcntl:

**Table 2-31. NU\_Fcntl Supported Arguments**

Command	Arguments
NU_SETFLAG	NU_BLOCK - Enable blocking on the socket. Note that sockets are blocking by default.
	NU_NO_BLOCK - Disable blocking on the socket.
NU_SET_ZC_MODE	NU_ZC_ENABLE - Enable Zero Copy mode on the socket. Note that zero copy is disabled on sockets by default.
	NU_ZC_DISABLE - Disable Zero Copy mode on the socket.

## Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- **NU\_NO\_ACTION**  
No action was processed by this function due to an invalid command parameter.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value.

## Example

```
INT      socketd; /* the socket descriptor */
STATUS status; /* return status of the NU_fcntl call */

. . .

/* Disable blocking on the socket */
status = NU_Fcntl(socketd, NU_SETFLAG, NU_NO_BLOCK);

/* if status equals NU_SUCCESS, the NU_Fcntl call was successful
   and the block flag is enabled */
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)



## NU\_FD\_Check (IPv4/IPv6)

This function is responsible for checking the specified bit in a bitmap to see if it is set. It is generally used in conjunction with [NU\\_Select](#) to check the bitmap returned. See [NU\\_Select \(IPv4/IPv6\)](#) for more information.

### Usage

```
INT NU_FD_Check (INT      socket,  
                 FD_SET *fd);
```

### Arguments

- `socket`  
Specifies the index of the bit to check.
- `fd`  
Pointer to the bitmap which will be checked.

### Return Values

- `NU_TRUE`  
The specified bit is set.
- `NU_FALSE`  
The specified bit was not set.

### Example

```
INT      socketd, i;  
FD_SET  readfs;  
.  
.  
.  
  
/* Make sure all bits are initially set to 0 */  
NU_FD_Init(&readfs);  
  
/* Set the bit for the sockets. */  
for (i = 0; i <= socketd; i++)  
{  
    NU_FD_Set(i, &readfs);  
}  
  
/* Now call NU_Select to see which, if any, have data. */  
if (NU_Select(socketd + 1, &readfs, NU_NULL,  
              NU_NULL, NU_NO_SUSPEND) == NU_SUCCESS)  
{  
    /* At least one of the sockets contained received data. Check all of  
     * them to find those that have data.  
     */  
    for (i = 0; i <= socketd; i++)  
    {  
        if (NU_FD_Check(i, &readfs) == NU_FALSE) continue;  
        .  
    }  
}
```

```
        .  
        .  
    }  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_FD\\_Set \(IPv4/IPv6\)](#)

[NU\\_FD\\_Init \(IPv4/IPv6\)](#)

[NU\\_FD\\_Reset \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

## NU\_FD\_Init (IPv4/IPv6)

This function is responsible for initializing the bitmaps used with the NU\_Select function. This call will initialize each bit in the bitmap to 0 and must be called before the routine NU\_FD\_Set.

### Usage

```
VOID NU_FD_Init (FD_SET *fd);
```

### Arguments

- fd  
A pointer to the bitmap that will be initialized.

### Example

In this example, the node is acting as a server.

```
INT      socketd, i;
FD_SET   readfs;

.
.
.

/* Make sure all bits are initially set to 0 */
NU_FD_Init(&readfs);

/* Set the bit for the first three sockets. */
for (i = socketd; i < 3; i++)
{
    NU_FD_Set(i, &readfs);
}

/* Now call NU_Select to see which, if any, have data. */
if (NU_Select(socketd + 1, &readfs, NU_NULL,
    NU_NULL, NU_NO_SUSPEND) == NU_SUCCESS)
{
    /* At least one of the sockets contained received data. Check all of
     * them to find those that have data.
     */
    for (i = 0; i <= socketd; i++)
    {
        if (NU_FD_Check(i, &readfs) == NU_FALSE) continue;
        .
        .
        .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_FD\\_Set \(IPv4/IPv6\)](#)

[NU\\_FD\\_Check \(IPv4/IPv6\)](#)

[NU\\_FD\\_Reset \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

## NU\_FD\_Reset (IPv4/IPv6)

This function is responsible for resetting one bit in the specified bitmap to 0. This service is generally used in conjunction with NU\_Select.

### Usage

```
VOID NU_FD_Reset (INT      socket,  
                  FD_SET *fd);
```

### Arguments

- socket  
Specifies the index of the bit to be reset.
- fd  
Pointer to the bitmap in which the bit will be reset.

### Example

In this example, the node is acting as a server.

```
INT      socketd, i;  
FD_SET readfs;  
  
. . .  
  
/* Now call NU_Select to see which, if any, have data. */  
if (NU_Select(socketd + 1, &readfs, NU_NULL,  
              NU_NULL, NU_NO_SUSPEND) == NU_SUCCESS)  
{  
    /* At least one of the sockets contained received data. Check all  
     * of them to find those that have data.  
     */  
    for (i = 0; i <= socketd; i++)  
    {  
        if (NU_FD_Check(i, &readfs) == NU_FALSE) continue;  
  
        /* We only want to receive data on this socket once,  
         * so reset the bit so we don't receive any more data on  
         * this socket.  
         */  
        NU_FD_Reset(i, &readfs);  
        .  
        .  
        .  
    }  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_FD\\_Set \(IPv4/IPv6\)](#)

[NU\\_FD\\_Check \(IPv4/IPv6\)](#)

[NU\\_FD\\_Init \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

## NU\_FD\_Set (IPv4/IPv6)

This function is responsible for setting the specified bit of a bitmap so the lower layer will inform the application when data is received on a particular socket. This service is generally used in conjunction with NU\_Select.

### Usage

```
VOID NU_FD_Set (INT      socket,  
               FD_SET *fd);
```

### Arguments

- socket  
The index of the bit to be set.
- fd  
Pointer to the bitmap in which the bit will be set.

### Example

```
INT      socketd, i;  
FD_SET  readfs;  
  
. . .  
  
/* Make sure all bits are initially set to 0 */  
NU_FD_Init(&readfs);  
  
/* Set the bit for the first three sockets. */  
for (i = 0; i <= socketd; i++)  
{  
    NU_FD_Set(i, &readfs);  
}  
  
/* Now call NU_Select to see which, if any, have data. */  
if (NU_Select(socketd + 1, &readfs, NU_NULL,  
             NU_NULL, NU_NO_SUSPEND) == NU_SUCCESS)  
{  
    /* At least one of the sockets contained received data.  
     * Check all of them to find those that have data.  
     */  
    for (i = socketd; i < 3; i++)  
    {  
        if (NU_FD_Check(i, &readfs) == NU_FALSE) continue;  
        .  
        .  
        .  
    }  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_FD\\_Reset \(IPv4/IPv6\)](#)

[NU\\_FD\\_Check \(IPv4/IPv6\)](#)

[NU\\_FD\\_Init \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)



## NU\_Find\_Next\_Route (IPv4/IPv6)

This function finds the route in the routing tree proceeding the provided route.

### Usage

```
ROUTE_ENTRY *NU_Find_Next_Route (const UINT8 *current_address,  
                                INT16         family);
```

### Arguments

- **current\_address**  
A pointer to the IP address of the route of which the next route is desired.
- **family**  
The family type of the address passed in, either NU\_FAMILY\_IP or NU\_FAMILY\_IP6.

### Return Values

- **ROUTE\_NODE**  
A pointer to the route.
- **NU\_NULL**  
There is no next route.

### Example

```
ROUTE_ENTRY current_route;  
UINT8 null_addr[] = {0,0,0,0};  
  
/* Get the first route in the tree. */  
current_route = NU_Find_Next_Route(null_addr, NU_FAMILY_IP);  
  
/* Traverse the entire routing table, gathering each valid route */  
while (current_route)  
{  
    /* Do something with the route. */  
    ...  
  
    /* Get a pointer to the next route */  
    current_route = NU_Find_Next_Route(current_route->rt_route_node  
->rt_ip_addr, NU_FAMILY_IP);  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Find\\_Next\\_Route\\_Entry](#)

[NU\\_Add\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)

## NU\_Find\_Next\_Route\_Entry

This function finds the next route in the routing table when there are multiple routes to the same destination.

### Usage

```
ROUTE_ENTRY *NU_Find_Next_Route_Entry (ROUTE_ENTRY *current_route,  
                                       INT16         family);
```

### Arguments

- **current route**  
A pointer to the route of which the next route is desired.
- **family**  
The family type of the route passed in, either NU\_FAMILY\_IP or NU\_FAMILY\_IP6.

### Return Values

- **ROUTE\_ENTRY**  
A pointer to the next route.
- **NU\_NULL**  
There is no next route.

### Example

```
ROUTE_ENTRY* *current_route;  
  
/* Get a pointer to the first route in the route table */  
current_route = NU_Find_Next_Route_Entry(NU_NULL, NU_FAMILY_IP);  
  
/* Check each route in the table. */  
while (current_route)  
{  
    /* Do something with the route entry. */  
    ...  
  
    /* Get a pointer to the next route. */  
    current_route = NU_Find_Next_Route_Entry(current_route, NU_FAMILY_IP);  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)

[NU\\_Get\\_Default\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

[NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)

[NU\\_Find\\_Next\\_Route\\_Entry](#)

## NU\_Find\_Route\_By\_Gateway (IPv4/IPv6)

Find a route through a specific gateway.

### Usage

```
ROUTE_ENTRY *NU_Find_Route_By_Gateway(const UINT8 *ip_dest,  
                                       const UINT8 *gw_addr,  
                                       INT16      family,  
                                       INT32      flags);
```

### Arguments

- **ip\_dest**  
The destination IP address of the route to find.
- **gw\_addr**  
The IP address of the gateway of the target route to find.
- **family**  
The family type of the route, either NU\_FAMILY\_IP or NU\_FAMILY\_IP6.
- **flags**  
Flags to apply to the route search.
  - RT\_HOST\_MATCH will find only a matching host route.
  - RT\_BEST\_METRIC returns the route with the best metric per RIPv2 or RIPv6.
  - RT\_OVERRIDE\_METRIC will return the route even if the metric is infinity.
  - RT\_OVERRIDE\_DV\_STATE will return the route even if the interface associated with the route is not UP and RUNNING.
  - RT\_OVERRIDE\_RT\_STATE will return the route even if the route is not UP.

### Return Values

- **ROUTE\_ENTRY**  
A pointer to the ROUTE\_ENTRY upon success.
- **NU\_NULL**  
Error code, if the route does not exist.

### Example

```
ROUTE_ENTRY *target_route;  
  
UINT8 ip_dest[] = {192, 168, 1, 100};  
UINT8 gw_addr[] = {192, 168, 1, 255};  
  
target_route = NU_Find_Route_By_Gateway(&ip_dest, &gw_addr, NU_FAMILY_IP,  
RT_HOST_MATCH);  
  
if (target_route)  
{
```

```
    ...  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)

[NU\\_Get\\_Default\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

[NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)

[NU\\_Find\\_Next\\_Route\\_Entry](#)

## NU\_Find\_Socket (IPv4/IPv6)

This function returns the socket descriptor associated with the local and foreign address and the local and foreign port number for the protocol specified.

### Usage

```
INT NU_Find_Socket (INT                protocol,  
                   const struct addr_struct *local_addr,  
                   const struct addr_struct *foreign_addr);
```

### Arguments

- `protocol`  
The protocol type of the socket to find, `NU_PROTO_TCP` or `NU_PROTO_UDP`.
- `local_addr`  
A pointer to the [addr\\_struct](#) data structure that holds the local address and port number of the local side.
- `foreign_addr`  
A pointer to the [addr\\_struct](#) data structure that holds the foreign address and port number of the foreign side.

### Return Values

- `socket`  
The socket descriptor (if successful).
- `NU_INVALID_PARM`  
One of the pointers is NULL.
- `NU_INVALID_PROTOCOL`  
The specified protocol is not valid.
- `NU_INVALID_SOCKET`  
There is no socket associated with the parameters provided.

### Example

```
INT socket;  
struct addr_struct local_addr, foreign_addr;  
  
/* Fill in a structure with the local address */  
local_addr.family = NU_FAMILY_IP;  
local_addr.port = 9;  
memcpy(&local_addr.id, IP_LOCAL_ADDR, IP_ADDR_LEN);  
  
/* Fill in a structure with the foreign address */  
foreign_addr.family = NU_FAMILY_IP;  
memcpy(&foreign_addr.id, IP_FOREIGN_ADDR, IP_ADDR_LEN);  
foreign_addr.port = 1000;
```

```
socket = NU_Find_Socket(NU_PROTO_TCP, &local_addr, &foreign_addr);  
  
if (socket >= 0)  
{  
    ...  
}
```

## Related Topics

[Net API Functions](#)[NU\\_Delete\\_Route \(IPv4\)](#)[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)[NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)[NU\\_Get\\_Default\\_Route \(IPv4\)](#)[NU\\_Add\\_Route \(IPv4\)](#)[NU\\_Add\\_Route6 \(IPv6\)](#)[NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)[NU\\_Find\\_Next\\_Route\\_Entry](#)

## NU\_Get\_Default\_Route (IPv4)

Get the default route for the system. There can be only one default route for each family type in the system.

### Usage

```
ROUTE_NODE *NU_Get_Default_Route (INT16 family);
```

### Arguments

- family  
The family type of the default route to retrieve, either NU\_FAMILY\_IP or NU\_FAMILY\_IP6.

### Return Values

- ROUTE\_NODE  
A Pointer to the default route.
- NU\_NULL  
No default route exists.

### Example

```
ROUTE_NODE *default_route;  
  
default_route = NU_Get_Default_Route (NU_FAMILY_IP);  
  
if (default_route)  
{  
    ...  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_Default\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

[NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)

[NU\\_Find\\_Next\\_Route\\_Entry](#)

## NU\_Free\_Host\_Entry (IPv4/IPv6)

This function frees the memory allocated by a previous call to [NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#) or [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#).

### Usage

```
STATUS NU_Free_Host_Entry (NU_HOSTENT *host_entry);
```

### Arguments

- `host_entry`  
A pointer to the memory allocated by the previous call to [NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#) or [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#). The structure `NU_HOSTENT` is defined in the [Net Data Structures](#) section.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- `NU_INVALID_POINTER`

The pointer to the memory is invalid.

### Example - IPv4

```
CHAR      ip_addr[] = {206,202,34,80};
NU_HOSTENT *hentry;
STATUS     status;

hentry = NU_Get_IP_Node_By_Addr(ip_addr, 4, NU_FAMILY_IP, &status);

/* Free the memory allocated for the host entry data structure
 * by the previous call.
 */
NU_Free_Host_Entry(hentry);
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)



## NU\_Get\_Default\_TTL (IPv4)

This function retrieves the default Time To Live value placed in the IP header of outgoing packets as specified by the macro `IP_TIME_TO_LIVE` located at *networking/ip.h*. This routine will never fail to retrieve a valid value.

### Usage

```
UINT8 NU_Get_Default_TTL (VOID);
```

### Return Values

- Default TTL value

### Example

```
UINT8    default_ttl;

/* Get the default TTL */
default_ttl = NU_Get_Default_TTL();
```

### Related Topics

[Net API Functions](#)

[NU\\_Set\\_Default\\_TTL \(IPv4\)](#)

## NU\_Get\_DHCP\_DUID (IPv4/IPv6)

This function gets the DUID (DHCP Unique Identifier) for the node for use with DHCP.

### Usage

```
STATUS NU_Get_DHCP_DUID (DHCP_DUID_STRUCT *duid_ptr);
```

### Arguments

- `duid_ptr`  
A pointer to the data structure that will be filled in by the function call.  
The [DHCP\\_DUID\\_STRUCT](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- An operating system-specific error code.  
Otherwise.

### Example

```
DHCP_DUID_STRUCT    duid_ptr;  
STATUS              status;  
  
/* Get the DUID for this node. */  
status = NU_Get_DHCP_DUID(&duid_ptr);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Get_DHCP_DUID.\n");  
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>	<a href="#">NU_Dhcp6 (IPv6)</a>
<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>	<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_Domain_Name (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## NU\_Get\_DHCP\_IAID (IPv4/IPv6)

This function retrieves the IAID (Identity Association Identifier) associated with an interface for use with DHCP.

### Usage

```
STATUS NU_Get_DHCP_IAID (UINT32 dev_index,  
                        UINT32 *iaid);
```

### Arguments

- `dev_index`  
The index number of the interface for which to retrieve the IAID.
- `iaid`  
A pointer to the memory in which to store the IAID of the interface.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - `NU_INVALID_PARM`  
One of the input parameters is invalid.

### Example

```
UINT32          iaid;  
DHCP6_CLIENT    ds_ptr;  
STATUS          status;  
  
/* Get the index associated with this interface name. */  
ds_ptr.dhcp6_dev_index = NU_IF_NameToIndex("eth0");  
  
/* Get the IA ID associated with this interface. */  
status = NU_Get_DHCP_IAID(ds_ptr.dhcp6_dev_index, &iaid);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Get_DHCP_IAID.\n");  
}
```

## Related Topics

[Net API Functions](#)

[DHCP6\\_Build\\_Option\\_Request\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_User\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Class\\_Option \(IPv6\)](#)

[DHCP6\\_Build\\_Vendor\\_Specific\\_Info\\_Option \(IPv6\)](#)

[NU\\_Dhcp6 \(IPv6\)](#)

[NU\\_Dhcp6\\_Shutdown \(IPv6\)](#)

[NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)

[DHCP6\\_Build\\_IA\\_NA\\_Option \(IPv6\)](#)

## NU\_Get\_DNS\_Servers (IPv4)

This function returns the list of IPv4 DNS servers being used by Nucleus NET.

### Usage

```
INT NU_Get_DNS_Servers (UINT8 *dest,  
                        INT     size);
```

### Arguments

- **dest**  
A pointer to a memory block where the list of IPv4 DNS server IP addresses will be copied.
- **size**  
The size of the block of memory pointed to by dest.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, the list of DNS servers will have been copied into the block of memory pointed to by dest. If dest is not large enough to hold the entire list, only as many addresses as will fit will be copied into dest. The number of addresses copied to dest will be returned upon success. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - **NU\_INVALID\_PARM**  
Either dest is a NULL pointer or size is less than four.

### Example

```
UINT8      dns1[] = {192,200,100,1};  
STATUS     status;  
UINT8      dns_list[5 * 4]; /* 5 ip addresses. */  
  
/* Add a new DNS server to the head of the list. */  
status = NU_Add_DNS_Server(dns1, DNS_ADD_TO_FRONT);  
  
if (status != NU_SUCCESS)  
{  
    printf("Could not add DNS Server.\n");  
}  
  
/* Now make sure the IP address for the server was added. */  
status = NU_Get_DNS_Servers(dns_list, sizeof(dns_list));  
  
if (status <= 0)  
{  
    printf("Could not obtain list of DNS Servers.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Delete\\_DNS\\_Server \(IPv4\)](#)

[NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)

[NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Add\\_DNS\\_Server \(IPv4\)](#)

## NU\_Get\_DNS\_Servers2 (IPv4/IPv6)

This function is used for IPv4 and IPv6 and replaces the function `NU_Get_DNS_Servers`. This function will return the list of IPv4 or IPv6 DNS servers that are being used by Nucleus NET.

### Usage

```
INT NU_Get_DNS_Servers2 (UINT8 *dest,  
                        INT size,  
                        INT16 family);
```

### Arguments

- **dest**  
A pointer to a memory block where the list of DNS server IP addresses will be copied.
- **size**  
The size of the block of memory pointed to by **dest**.
- **family**  
The family type of the DNS Servers to retrieve. Either `NU_FAMILY_IP` for IPv4 DNS Servers or `NU_FAMILY_IP6` for IPv6 DNS Servers.

### Return Values

Upon successful completion, `NU_Get_DNS_Servers2()` will return the number of addresses stored in the buffer pointed to by **dest**, and the list of IPv4 or IPv6 DNS servers will have been copied into the block of memory pointed to by **dest**. If **dest** is not large enough to hold the entire list, only as many addresses as will fit will be copied into **dest**. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- `NU_INVALID_PARM`  
**dest** is a NULL pointer, **size** is less than 16, or the family specified is invalid.

### Example - IPv4

```
UINT8      dns1[] = {192,200,100,1};  
STATUS     status;  
UINT8      dns_list[5 * IP_ADDR_LEN]; /* 5 ip addresses. */  
  
/* Add a new IPv4 DNS server to the head of the list. */  
status = NU_Add_DNS_Server2(dns1, DNS_ADD_TO_FRONT, NU_FAMILY_IP);  
  
if (status != NU_SUCCESS)  
{  
    printf("Could not add DNS Server.\n");  
}  
  
/* Now make sure the IPv4 address for the server was added. */  
status = NU_Get_DNS_Servers2(dns_list, sizeof(dns_list),  
                             NU_FAMILY_IP);
```

```
if (status <= 0)
{
    printf("Could not obtain list of DNS Servers.\n");
}
```

### Example - IPv6

```
UINT8          dns1[] =
    {0xfe,0x80,0,0,0,0,0,0,0,0,0,0,0xa,0xb,0xff,0x2,0xc,0xd};
STATUS          status;
UINT8          dns_list[5 * IP6_ADDR_LEN]; /* 5 ip addresses. */

/* Add a new IPv6 DNS server to the head of the list. */
status = NU_Add_DNS_Server2(dns1, DNS_ADD_TO_FRONT,
                           NU_FAMILY_IP6);

if (status != NU_SUCCESS)
{
    printf("Could not add DNS Server.\n");
}

/* Now make sure the IPv6 address for the server was added. */
status = NU_Get_DNS_Servers2(dns_list, sizeof(dns_list),
                             NU_FAMILY_IP6);

if (status <= 0)
{
    printf("Could not obtain list of DNS Servers.\n");
}
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Add_DNS_Server2 (IPv4/IPv6)</a>
<a href="#">NU_Delete_DNS_Server (IPv4)</a>	<a href="#">NU_Delete_DNS_Server2 (IPv4/IPv6)</a>
<a href="#">NU_Free_Host_Entry (IPv4/IPv6)</a>	<a href="#">NU_Get_DNS_Servers (IPv4)</a>
<a href="#">NU_Add_DNS_Server (IPv4)</a>	



## NU\_Get\_Domain\_Name (IPv4/IPv6)

This function returns the name of the local domain. The domain name is initially specified by the definition of DOMAINNAME in *networking/net\_cfg.h*. During initialization DOMAINNAME is copied into the global variable SCK\_Domain\_Name.

### Usage

```
INT NU_Get_Domain_Name (CHAR *name,  
                        INT  name_length);
```

### Arguments

- **name**  
A pointer to the location where the domain name will be copied.
- **name\_length**  
Size of the memory location pointed to by name.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, the name of the domain will be copied into the memory location pointed to by the name parameter. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
There is a problem with one of the parameters. It is likely that name is a NULL pointer or the name\_length is either zero or too small to hold the complete domain name.

### Example

```
CHAR    dname[64];  
STATUS  status;  
  
status = NU_Get_Domain_Name(dname, 64);  
  
if (status != NU_SUCCESS)  
{  
    printf("Failed to set the Domain Name.\n");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)

## NU\_Get\_Host\_By\_Addr (IPv4/IPv6)

This function returns a pointer to a host structure based on the IP address specified.

### Usage

```
STATUS NU_Get_Host_By_Addr (CHAR      *addr,  
                           INT        len,  
                           INT        type,  
                           NU_HOSTENT *host_entry);
```

### Arguments

- **addr**  
This parameter specifies the IP address of the host to be resolved.
- **len**  
The length of addr. A value of 4 should always be used for an IPv4 address. A value of 16 should always be used for an IPv6 address.
- **type**  
The type of address in parameter one – either NU\_FAMILY\_IP to resolve an IPv4 address or NU\_FAMILY\_IP6 to resolve an IPv6 address.
- **host\_entry**  
A pointer to the [NU\\_HOSTENT](#) structure filled in by the DNS Server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, and the structure pointed to by the host will be filled in with the host information. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_PARM**  
One of the parameters is a NULL pointer or the family specified is invalid.
  - **NU\_NO\_DNS\_SERVER**  
There are no DNS servers registered with Nucleus NET.
  - **NU\_DNS\_ERROR**  
General DNS lookup failure.
  - **NU\_NO\_MEMORY**  
Memory allocation failed; therefore, the host was not added.
  - **NU\_INVALID\_LABEL**  
The label exceeded the maximum DNS length.

## Description

First, the local “hosts list” is searched for the specified host. If not found in the local cache, the Domain Name System (DNS) is used to resolve the host via the network. Although this function is compatible with both IPv4 and IPv6, it is suggested that the application use the new API function `NU_Get_IP_Node_By_Addr` to resolve DNS records by address.

## Example - IPv4

```
CHAR          ip_addr[] = {206,202,34,80};
NU_HOSTENT    hentry;

if (NU_Get_Host_By_Addr(ip_addr, 4, NU_FAMILY_IP,
    &hentry) != NU_SUCCESS)
{
    printf("Could not resolve host name.\n");
}
```

## Example - IPv6

```
CHAR          ip_addr[] =
    {0x3f, 0xfe, 2, 0, 0, 8, 0, 0, 0, 0, 0, 0, 1, 9, 1};
NU_HOSTENT    hentry;

if (NU_Get_Host_By_Addr(ip_addr, 16, NU_FAMILY_IP6,
    &hentry) != NU_SUCCESS)
{
    printf("Could not resolve host name.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

## NU\_Get\_Host\_By\_Name (IPv4)

This function returns a pointer to a host structure based on the name specified.

### Usage

```
STATUS NU_Get_Host_By_Name (CHAR          *name,  
                           NU_HOSTENT *host_entry);
```

### Arguments

- name  
Pointer to the name of the host to be resolved.
- host\_entry  
Pointer to the [NU\\_HOSTENT](#) filled in by the DNS Server.

### Return Values

- NU\_SUCCESS  
Upon successful completion, and the structure pointed to by host\_entry will be filled in with the host information. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - NU\_INVALID\_PARM  
One of the parameters is a NULL pointer.
  - NU\_NO\_DNS\_SERVER  
There are no DNS servers registered with Nucleus NET.
  - NU\_DNS\_ERROR  
General DNS lookup failure.
  - NU\_INVALID\_LABEL  
Label exceeds the maximum DNS length.

### Description

First the local “hosts list” is searched for the specified host. If not found in the local cache, the Domain Name System (DNS) is used to resolve the host via the network.

### Example

```
CHAR          foreignname[] = "test_server";  
NU_HOSTENT    hentry;  
struct addr_struct servaddr;  
  
/* Retrieve the IP address of the server */  
if (NU_Get_Host_By_Name(foreignname, &hentry) != NU_SUCCESS)
```

```
{  
    printf("Cannot resolve host address().\n");  
}  
  
/* Fill in a structure with the server address */  
memcpy(&servaddr.id, hentry.h_addr, hentry.h_length);  
  
servaddr.family    = hentry.h_addrtype;  
servaddr.port      = 7;
```

## Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

## NU\_Get\_Host\_MX

This function resolves the MX host name associated with a domain name.

### Usage

```
STATUS NU_Get_Host_MX (CHAR      *name,  
                      DNS_MX_RR *mx_records,  
                      UINT16     *record_count);
```

### Arguments

- **name**  
Pointer to the host name to resolve into one or more MX records.
- **mx\_records**  
Pointer to the memory in which to store the records.
- **record\_count**  
Upon calling the routine, this points to the total number of DNS\_MX\_RR's that will fit in the user buffer.

### Return Values

- **NU\_SUCCESS**  
Successful operation, upon return \*record\_count contains the number of records filled into the buffer.
- **NU\_NO\_MEMORY**  
No memory.
- **NU\_DNS\_ERROR**  
DNS could not resolve.
- **NU\_NO\_DNS\_SERVER**  
No DNS server exists.

### Description

Upon successful completion of the routine, you can then resolve those host names into IP addresses using [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#).

### Example

```
STATUS status;  
DNS_MX_RR mx_records;  
UINT16 record_count = 1;  
  
status = NU_Get_Host_MX("mentor.com", &mx_records, &record_count);
```

## Related Topics

[Net API Functions](#)

[NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)

[NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

[NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)

[NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)

## NU\_Get\_Host\_Name (IPv4/IPv6)

This function returns the name of the local host.

### Usage

```
INT NU_Get_Host_Name (CHAR *name,  
                     INT  name_length);
```

### Arguments

- **name**  
A pointer to the location where the host name will be copied.
- **name\_length**  
Size of the memory location pointed to by name.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, the name of the host will be copied into the memory location pointed to by the name parameter. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - **NU\_INVALID\_PARM**  
There is a problem with one of the parameters. It is likely that name is a null pointer or the name\_length is either 0 or too small to hold the complete hostname.

### Description

The host name is initially specified by the definition of HOSTNAME found in *networking/net\_cfg.h*. During initialization, HOSTNAME is copied into the global variable SCK\_Host\_Name.

### Example

```
CHAR      hname[64];  
STATUS    status;  
  
status = NU_Get_Host_Name(hname, 64);  
  
if (status != NU_SUCCESS)  
{  
    printf("Failed to get the host name.\n");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Set\\_Host\\_Name \(IPv4/IPv6\)](#)



## NU\_Get\_IP\_Forwarding (IPv4/IPv6)

This function determines whether IP Forwarding is enabled or disabled in the system.

### Usage

```
UINT8 NU_Get_IP_Forwarding (VOID);
```

### Return Values

- NU\_TRUE  
IP forwarding is enabled.
- NU\_FALSE  
IP forwarding is disabled.

### Description

By default, if the macro `INCLUDE_IP_FORWARDING` located at *networking/net\_cfg.h* is set to `NU_TRUE`, IP forwarding is enabled. If this macro is set to `NU_FALSE`, IP forwarding will always be disabled.

### Example

```
UINT8    ip_forwarding;  
  
/* Determine whether IP Forwarding is enabled */  
ip_forwarding = NU_Get_IP_Forwarding();
```

### Related Topics

[Net API Functions](#)

[NU\\_Set\\_IP\\_Forwarding \(IPv4/IPv6\)](#)

## NU\_Get\_IP\_Node\_By\_Addr (IPv4/IPv6)

This function is used by both IPv4 and IPv6 and replaces the function `NU_Get_Host_By_Addr`. This function returns a pointer to a host structure based on the IP address specified.

### Usage

```
NU_HOSTENT *NU_Get_IP_Node_By_Addr (CHAR    *addr,  
                                     INT      len,  
                                     INT      type,  
                                     STATUS    *error_num);
```

### Arguments

- `addr`  
The IP address of the host to be resolved.
- `len`  
The length of `addr`. A value of 4 should always be used for an IPv4 address. A value of 16 should always be used for an IPv6 address.
- `type`  
The type of address in parameter one - either `NU_FAMILY_IP` to resolve an IPv4 address or `NU_FAMILY_IP6` to resolve an IPv6 address.
- `error_num`  
A pointer to the status of this call.

### Return Values

- `NU_SUCCESS`  
Upon successful completion of this routine, a pointer to the filled in `NU_HOSTENT` data structure will be returned to the application and the parameter `error_num` will be set to `NU_SUCCESS`. The `NU_HOSTENT` data structure is defined in the [Net Data Structures](#) section.

Otherwise, a pointer to `NU_NULL` will be returned and the value of `error_num` will be set to an operating system-specific error code or one of the following Nucleus NET error codes:

- `NU_INVALID_PARM`  
One of the parameters is a NULL pointer or the family specified is invalid.
- `NU_NO_DNS_SERVER`  
There are no DNS servers registered with Nucleus NET.
- `NU_DNS_ERROR`  
General DNS lookup failure.
- `NU_INVALID_LABEL`  
The label exceeded the maximum DNS length.

- NU\_NOT\_A\_HOST

The attempt to resolve timed out.

## Description

First the local “hosts list” is searched for the specified host. If not found in the local cache, the Domain Name System (DNS) is used to resolve the host via the network. mDNS will be used for the query if there are no DNS servers configured to service the request and mDNS has been included in the *.metadata* file at compilation time. mDNS transmits the request over UDP port 5353 to the multicast IP address 224.0.0.251 for IPv4 and ff02::fb for IPv6. Note that the function [NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#) must be used to free the memory allocated for the [NU\\_HOSTENT](#) data structure returned by this function.

## Example - IPv4

```
CHAR          ip_addr[] = {206,202,34,80};
NU_HOSTENT    *hentry;
STATUS        status;

hentry = NU_Get_IP_Node_By_Addr(ip_addr, 4, NU_FAMILY_IP,
                                &status);

/* Free the memory allocated for the host entry data structure
 * by the previous call.
 */
NU_Free_Host_Entry(hentry);
```

## Example - IPv6

```
CHAR          ip_addr[] =
    {0x3f, 0xfe, 2, 0, 8, 0, 0, 0, 0, 0, 0, 1, 9, 1};
NU_HOSTENT    *hentry;
STATUS        status;

hentry = NU_Get_IP_Node_By_Addr(ip_addr, 16, NU_FAMILY_IP6,
                                &hentry);

/* Free the memory allocated for the host entry data structure
 * by the previous call.
 */
NU_Free_Host_Entry(hentry);
```

## Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)

## NU\_Get\_IP\_Node\_By\_Name (IPv4/IPv6)

This function is used for IPv4 and IPv6 and replaces the function [NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#). This function returns a pointer to a host structure based on the name specified.

### Usage

```
NU_HOSTENT *NU_Get_IP_Node_By_Name (CHAR    *name,  
                                     INT16   family,  
                                     INT16   flags,  
                                     STATUS  *error_num);
```

### Arguments

- **name**  
The name of the host to resolve.
- **family**  
The family type of the host to resolve – either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6.
- **flags**

Flags to indicate the following:

DNS\_V4MAPPED

If specified with a family type of NU\_FAMILY\_IP6 and no AAAA records are found, a query is made for A records and any found are returned as IPv4-Mapped IPv6 addresses. This flag is ignored unless the family type is NU\_FAMILY\_IP6.

DNS\_ALL | DNS\_V4MAPPED

The caller wants all addresses; IPv6 and IPv4-Mapped IPv6. This flag is ignored unless the family type is NU\_FAMILY\_IP6 and the flag DNS\_V4MAPPED is also set.

DNS\_ADDRCONFIG

A query for AAAA records should occur only if the node has at least one IPv6 source address configured and a query for A records should occur only if the node has at least one IPv4 source address configured.

DNS\_DEFAULT

(DNS\_V4MAPPED|DNS\_ADDRCONFIG)

- **error\_num**  
A pointer to the status of the call.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, NU\_Get\_IP\_Node\_By\_Name() returns a pointer to the [NU\\_HOSTENT](#) data structure containing the resolved DNS records, and the parameter error\_num is set to NU\_SUCCESS.

Upon failure, the [NU\\_HOSTENT](#) data structure is set to NU\_NULL, and the value of error\_num is set to an operating system-specific error code or one of the following Nucleus NET error codes:

- **NU\_INVALID\_PARM**  
 One of the parameters is a NULL pointer.
- **NU\_NO\_DNS\_SERVER**  
 There are no DNS servers registered with Nucleus NET.
- **NU\_DNS\_ERROR**  
 No records of the specified type could be found, because there is no corresponding IP address of the specified type on a device in the system.

## Description

First the local “hosts list” is searched for the specified host. If not found in the local cache, the Domain Name System (DNS) is used to resolve the host via the network. mDNS will be used for the query if the domain of the target is “.local” or if there are no DNS servers configured to service the request and mDNS has been included in the *.metadata* file at compilation time. mDNS transmits the request over UDP port 5353 to the multicast IP address 224.0.0.251 for IPv4 and ff02::fb for IPv6. Note that a call to `NU_Free_Host_Entry` is required to free the memory associated with Host Entry data structure returned to the application.

## Example - IPv4

```

CHAR          name[] = "model";
NU_HOSTENT    *hentry;
STATUS        status;
UINT8        *addr_list;
INT           i;
UINT8        *addr;
UINT8        null_addr[] = {0, 0, 0, 0};

/* Resolve the IPv4 address of the name */
hentry = NU_Get_IP_Node_By_Name(name, NU_FAMILY_IP, 0, &status);

if (status == NU_SUCCESS)
{
    /* Get a pointer to the head of the address list. */
    addr_list = *hentry->h_addr_list;

    /* Traverse the entire address list. */
    for (i = 0; ; i += IP_ADDR_LEN)
    {
        /* Get a pointer to this address in the list. */
        addr = &addr_list[i];

        . . .

        /* If this address is all zeros, we are at the end
         * of the list.
         */
    }
}

```

---

```

        if (memcmp(addr, null_addr, IP_ADDR_LEN) == 0)
            break;
    }

    /* Free the memory associated with the host entry returned */
    NU_Free_Host_Entry(hentry);
}

```

### Example - IPv6

```

CHAR   name[] = "ip6.atinucleus.com";
NU_HOSTENT *hentry;
STATUS status;
UINT8   *addr_list;
INT      i;
UINT8   *addr;
UINT8   null_addr[] =
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

/* Return IPv6 and IPv4-Mapped IPv6 addresses associated with
 * the name.
 */
hentry = NU_Get_IP_Node_By_Name(name, NU_FAMILY_IP6,
                                DNS_ALL | DNS_V4MAPPED, &status);

if (status == NU_SUCCESS)
{
    /* Get a pointer to the head of the address list. */
    addr_list = *hentry->h_addr_list;

    /* Traverse the entire address list. */
    for (i = 0; ; i += IP6_ADDR_LEN)
    {
        /* Get a pointer to this address in the list. */
        addr = &addr_list[i];

        . . .

        /* If this address is all zeros, we are at the end
         * of the list.
         */
        if (memcmp(addr, null_addr, IP6_ADDR_LEN) == 0)
            break;
    }

    /* Free the memory associated with the host entry returned */
    NU_Free_Host_Entry(hentry);
}

/* Return IPv4-Mapped IPv6 addresses associated with the name
 * only if no IPv6 addresses are found.
 */
hentry = NU_Get_IP_Node_By_Name(name, NU_FAMILY_IP6, DNS_V4MAPPED,
                                &status);

if (status == NU_SUCCESS)
{
    /* Get a pointer to the head of the address list. */

```

```

addr_list = *hentry->h_addr_list;

/* Traverse the entire address list. */
for (i = 0; ; i += IP6_ADDR_LEN)
{
    /* Get a pointer to this address in the list. */
    addr = &addr_list[i];

    . . .

    /* If this address is all zeros, we are at the end
     * of the list.
     */
    if (memcmp(addr, null_addr, IP6_ADDR_LEN) == 0)
        break;
}

/* Free the memory associated with the host entry returned */
NU_Free_Host_Entry(hentry);
}

/* Return IPv6 addresses only if there is a device in the system
 * with a native IPv6 address.
 */
hentry = NU_Get_IP_Node_By_Name(name, NU_FAMILY_IP6, DNS_ADDRCONFIG,
                                &status);

if (status == NU_SUCCESS)
{
    /* Get a pointer to the head of the address list. */
    addr_list = *hentry->h_addr_list;

    /* Traverse the entire address list. */
    for (i = 0; ; i += IP6_ADDR_LEN)
    {
        /* Get a pointer to this address in the list. */
        addr = &addr_list[i];

        . . .

        /* If this address is all zeros, we are at the end
         * of the list.
         */
        if (memcmp(addr, null_addr, IP6_ADDR_LEN) == 0)
            break;
    }

    /* Free the memory associated with the host entry returned */
    NU_Free_Host_Entry(hentry);
}

```

**Related Topics**[Net API Functions](#)[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)[NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)



## NU\_Get\_Link\_Local\_Addr (IPv6)

This function retrieves the link-local address associated with an interface.

### Usage

```
STATUS NU_Get_Link_Local_Addr (UINT32 dev_index,  
                               UINT8  *addr_ptr);
```

### Arguments

- **dev\_index**  
The index of the interface for which to retrieve the link-local address.
- **addr\_ptr**  
A pointer to 16 bytes of memory that the function will fill in with the link-local address associated with the interface.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, `addr_ptr` contains the link-local address associated with the interface. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - **NU\_INVALID\_PARM**  
One of the input parameters is invalid.

### Example

```
UINT32  dev_index;  
UINT8   addr_ptr[IP6_ADDR_LEN];  
STATUS  status;  
  
/* Get the index associated with this interface name. */  
dev_index = NU_IF_NameToIndex("eth0");  
  
/* Get the link-local address associated with the interface. */  
status = NU_Get_Link_Local_Addr(dev_index, addr_ptr);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Get_Link_Local_Addr.\n");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

## NU\_Get\_Peer\_Name (IPv4/IPv6)

This function is responsible for returning the remote endpoint of a specified connection. A server may call this function to determine the address of the client to which it is connected.

### Usage

```
STATUS NU_Get_Peer_Name (INT socketd,  
                        struct sockaddr_struct *peer,  
                        INT16 *addr_length);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **peer**  
A pointer to a structure containing the following information about the remote peer:
  - **ip\_num**  
Contains the IP address of the peer.
  - **port\_num**  
Contains the port number of the peer.
  - **family**  
Contains the family type:
    - NU\_FAMILY\_IP for IPv4
    - NU\_FAMILY\_IP6 for IPv6
- **addr\_length**  
Pointer to the length of the peer structure, on return.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket descriptor was invalid.
  - **NU\_NOT\_CONNECTED**  
The specified socket descriptor did not have a client connected to it.
  - **NU\_INVALID\_PARM**  
The address length is invalid.

## Example

```
STATUS status;          /* status of Get Peer Name call */
INT     socketd;         /* the socket descriptor */
struct sockaddr_struct peer; /* holds the client remote address */
int     addr_length;     /* length of the peer structure */
.
.
status = NU_Get_Peer_Name(socketd, &peer, &addr_length);

/* if status is equal to NU_SUCCESS, then peer contains the
   remote address of the client to which the server is connected
   for the specified socket descriptor and addr_length contains
   the length of the peer structure */
```

## Related Topics

[Net API Functions](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

## NU\_Get\_PMTU (IPv4/IPv6)

This function is responsible for returning the Path MTU for transmitting data to the destination.

### Usage

```
UINT32 NU_Get_PMTU (UINT8  *src_addr,  
                   UINT8  *dest_addr,  
                   INT16  family,  
                   STATUS *status);
```

### Arguments

- `src_addr`  
The source address from which data is being transmitted.
- `dest_addr`  
The destination address to which data is being transmitted.
- `family`  
The family type of the source and destination addresses: `NU_FAMILY_IP` for IPv4, or `NU_FAMILY_IP6` for IPv6.
- `status`  
Pointer to the return status of the function.

### Return Values

- `NU_SUCCESS`  
Upon successful completion, `NU_Get_PMTU()` returns the Path MTU to the destination. Otherwise, the function returns 0, and status contains an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_PARM`  
One of the input parameters is invalid.
  - `NU_NO_ROUTE_TO_HOST`  
No route exists to the destination.

### Example

```
UINT32  pmtu;  
struct addr_struct  server_addr;  
  
/* Initialize Nucleus NET and the networking interface(s) */  
. . .  
  
/* Initiate a connection with another node */  
. . .  
  
/* Get the PMTU for the connection */  
pmtu = NU_Get_PMTU (local_addr, server_addr.id, NU_FAMILY_IP,
```

&status);

## Related Topics

[Net API Functions](#)

[NU\\_Update\\_Route \(IPv4/IPv6\)](#)

## NU\_Get\_Reasm\_Max\_Size (IPv4)

This function retrieves the Maximum Reassembly Size for an interface. Upon initialization of the interface, the Maximum Reassembly Size is initialized to the link MTU for the interface.

### Usage

```
STATUS NU_Get_Reasm_Max_Size (CHAR    *dev_name,  
                             UINT32  *reasm_max_size);
```

### Arguments

- **dev\_name**  
A pointer to the name of the device for which to retrieve the Maximum Reassembly Size.
- **reasm\_max\_size**  
Pointer to the Maximum Reassembly Size for the device.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - **NU\_INVALID\_PARM**  
One of the input parameters is invalid.

### Example

```
UINT32    max_reasm_size;  
  
/* Get the maximum reassembly size */  
if (NU_Get_Reasm_Max_Size("eth0",  
    &max_reasm_size) != NU_SUCCESS)  
{  
    printf("Could not get the max reassembly size.\n");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Set\\_Reasm\\_Max\\_Size \(IPv4\)](#)

## NU\_Get\_Sock\_Name (IPv4/IPv6)

This function is responsible for returning the local endpoint of a specified connection. A server may call this function to determine the IP address and port number of the local end of a socket.

### Usage

```
STATUS NU_Get_Sock_Name (INT          socketd,
                        struct sockaddr_struct *localaddr,
                        INT16          *addr_length);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **localaddr**  
A pointer to a structure containing the local address.
- **addr\_length**  
Return pointer for the length of the localaddr structure.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket descriptor was invalid.
  - **NU\_INVALID\_PARM**  
The address length is invalid.
  - **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.

### Example

```
STATUS status;          /* status of Get Sock Name call */
INT     socketd;        /* the socket descriptor */
struct sockaddr_struct localaddr; /* local socket address */
int     addr_length;    /* length of localaddr structure */
.
.
status = NU_Get_Sock_Name(socketd, &localaddr, &addr_length);

/* if status is equal to NU_SUCCESS, then localaddr contains the
   local addresses, IP and port, of the socket to which the
   server is connected for the specified socket descriptor and
   addr_length contains the length of the localaddr structure */
```

## Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Domain\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_MX](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Set\\_Host\\_Name \(IPv4/IPv6\)](#)



## NU\_Getsockopt (IPv4/IPv6)

This function is responsible for returning the status of an option for a specified socket. The NU\_Setsockopt service can be used to set socket options.

### Usage

```
STATUS NU_Getsockopt (INT    socketd,  
                     INT    level,  
                     INT    optname,  
                     VOID    *optval,  
                     INT    *optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **level**  
Specifies the protocol level. Valid entries for this parameter are SOL\_SOCKET, IPPROTO\_IP, IPPROTO\_IPV6, IPPROTO\_TCP, and IPPROTO\_UDP.

[Table 2-32](#), [Table 2-33](#), [Table 2-34](#), [Table 2-35](#), and [Table 2-36](#) list the levels and the associated options that are currently supported by NU\_Getsockopt.

**Table 2-32. SOL\_SOCKET Level Options**

SOL_SOCKET Level	Family	Option
SO_BROADCAST	IPv4	Check the broadcast status of a socket. When a socket is created the ability to send broadcasts is enabled by default. Upon a successful return the SO_BROADCAST bit will be set in the optval parameter.
SO_LINGER	IPv4/IPv6	Determine whether the linger option is enabled on the socket.
SO_REUSEADDR	IPv4/IPv6	Determine whether the option to reuse a port number for a different IP address is set on the socket.
SO_RCVBUF	IPv4/IPv6	Gets the TCP Window Size for the local side of the connection on a socket. This socket option has been deprecated in favor of TCP_RCV_WINDOWSIZE in order to take advantage of the larger window size made possible by the use of the TCP Window Scale option.

**Table 2-33. IPPROTO\_IP Level Options**

IPPROTO_IP Level	Family	Option
IP_MULTICAST_TTL	IPv4	Check the TTL (Time To Live) value that is used for multicast packets that are sent using this socket. The default TTL is 1. This keeps routers from forwarding the multicast datagrams beyond the local network. Upon successful completion, the optval parameter will contain the multicast TTL for the socket.
IP_MULTICAST_IF	IPv4	Check which interface has been set as the interface for sending multicast datagrams. The IP address of the interface is returned in the optval parameter as a 4-byte array.
IP_BROADCAST_IF	IPv4	Check which interface has been set as the interface for sending broadcast datagrams. The IP address of the interface is returned in the optval parameter as a 4-byte array.
IP_RECVIFADDR	IPv4	Check if the receive interface address option has been enabled. This option enables the storing of the receive interface upon calls to NU_Recv_From(). After a call to NU_Recv_From(), a call to NU_Recv_IF_Addr() can be made in order to obtain the IP address of the interface which received the datagram. This is useful when receiving multicast and broadcast packets that do not have a specific destination IP address.
IP_HDRINCL	IPv4	This option is only valid for IPv4 RAW sockets. When set, this option specifies that the application layer is responsible for filling in the IPv4 header. When clear, the application will only pass data to the stack, which will then append an IPv4 header. This option is only valid when using the raw IP services.
IP_TTL	IPv4	Retrieve the IPv4 Time To Live for the specified socket.
IP_TOS	IPv4	Retrieve the Type of Service/Differentiated Service Code Point for the specified IPv4 socket.

**Table 2-34. IPPROTO\_TCP Level Options**

<b>IPPROTO_TCP Level</b>	<b>Family</b>	<b>Option</b>
SO_KEEPAIVE	IPv4/IPv6	This option determines whether TCP Keep-Alive has been enabled on the socket.
TCP_CFG_DSACK	IPv4/IPv6	Get the value of TCP DSACK support on the socket.
TCP_CFG_SACK	IPv4/IPv6	Get the value of TCP SACK support on the socket.
TCP_CONGESTION_CTRL	IPv4/IPv6	Get the value of the TCP Congestion Control algorithm on the socket.
TCP_DELAY_ACK	IPv4/IPv6	Get the amount of time to delay before ACKing incoming data.
TCP_FIRST_PROBE_TIMEOUT	IPv4/IPv6	Get the amount of time to delay before transmitting the initial Zero Window probe.
TCP_FIRST_RTO	IPv4/IPv6	Get the amount of time to delay before sending the first retransmission of unACKed data.
TCP_KEEPAIVE_R2	IPv4/IPv6	Get the maximum number of Keep-Alive packets to transmit before closing the connection.
TCP_KEEPAIVE_WAIT	IPv4/IPv6	Get the amount of time to remain idle on a connection before invoking the Keep-Alive mechanism.
TCP_MAX_PROBES	IPv4/IPv6	Get the maximum number of Zero Window probes to transmit before closing the connection.
TCP_MAX_R2	IPv4/IPv6	Get the maximum number of retransmissions of unACKed data.
TCP_MAX_RTO	IPv4/IPv6	Get the maximum amount of time to delay between successive retransmissions of unACKed data.
TCP_MAX_SYN_R2	IPv4/IPv6	Get the maximum number of retransmissions of unACKed SYNs.
TCP_MSL	IPv4/IPv6	Get the maximum amount of time to wait before reusing the port / IP address combination.

**Table 2-34. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_NODELAY	IPv4/IPv6	TCP sockets, by default, utilize the Nagle Algorithm. The Nagle Algorithm attempts to merge contiguous small TCP packets into a single large TCP packet. It does this by delaying the transmission of small packets. This better utilizes the network bandwidth. However, it is often desirable for data to be transmitted immediately. The TCP_NODELAY option can be used to retrieve the current disposition of the Nagle Algorithm. A value of 1 indicates no delay, for example, the Nagle Algorithm is off. A value of 0 indicates small packets will be delayed, for example, the Nagle Algorithm is on.
TCP_PROBE_TIMEOUT	IPv4/IPv6	Get the maximum amount of time to delay between successive Zero Window probes.
TCP_RCV_WINDOWSIZE	IPv4/IPv6	Get the value of the local receive window for the socket.
TCP_SND_WINDOWSIZE	IPv4/IPv6	Get the value of the foreign receive window for the socket.
TCP_TIMESTAMP	IPv4/IPv6	Get the value of the Timestamp option for the socket.
TCP_WINDOWSCALE	IPv4/IPv6	Get the value of TCP Window Scale option support on the socket. If support is enabled, this socket option also returns the value of the shift count for the option.

**Table 2-35. IPPROTO\_UDP Level Options**

IPPROTO_UDP Level	Family	Option
UDP_NOCHECKSUM	IPv4	Determine whether the UDP_NOCHECKSUM option is enabled. A value of zero indicates the option is disabled. A non-zero value indicates the option is enabled.

**Table 2-36. IPPROTO\_IPV6 Level**

IPPROTO_IPV6 Level	Family	Option
IPV6_CHECKSUM	IPv6	Retrieves the value of the socket option for enabling or disabling the socket option for the IP layer to compute the checksum for outgoing IPv6 RAW packets and verify the checksum for incoming IPv6 RAW packets on the socket.
IPV6_DSTOPTS	IPv6	Retrieves the value of the Destination Options Sticky Option that was previously set by the application.
IPV6_HOPOPTS	IPv6	Retrieves the value of the Hop-By-Hop Options Sticky Option that was previously set by the application.
IPV6_MULTICAST_HOPS	IPv6	Retrieves the value of the hop limit used for outgoing multicast packets on the socket.
IPV6_MULTICAST_IF	IPv6	Retrieves the value of the interface used for outgoing multicast packets on the socket.
IPV6_NEXTHOP	IPv6	Retrieves the value of the Next-Hop Sticky Option that was previously set by the application.
IPV6_PKTINFO	IPv6	Retrieves the value of the source address and outgoing interface index Sticky Option that was previously set by the application.
IPV6_RECVDSOPTS	IPv6	Returns the value of the socket option to receive any Destination Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPLIMIT	IPv6	Returns the value of the socket option to receive the Hop Limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPOPTS	IPv6	Returns the value of the socket option to receive any Hop-By-Hop Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

**Table 2-36. IPPROTO\_IPV6 Level (cont.)**

IPPROTO_IPV6 Level	Family	Option
IPV6_RECVPKTINFO	IPv6	Returns the value of the socket option to receive the receive interface and destination address of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVRTHDR	IPv6	Returns the value of the socket option to receive the Routing Header for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVTCLASS	IPv6	Retrieves the value of the socket option for enabling or disabling the socket option to return the Traffic Class of the most recently received IPv6 packet on a socket. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RTHDR	IPv6	Retrieves the value of the Routing Header Sticky Option that was previously set by the application.
IPV6_RTHDRDSTOPTS	IPv6	Retrieves the value of the Destination Options Sticky Option for Destination Options that precede the Routing Header that was previously set by the application.
IPV6_TCLASS	IPv6	Retrieves the value of the Traffic Class Sticky Option that was previously set by the application.
IPV6_V6ONLY	IPv6	Retrieves the value of the socket option to receive IPv6 packets only on a socket of family type NU_FAMILY_IP6.
IPV6_UNICAST_HOPS	IPv6	Retrieves the value of the hop limit used in outgoing unicast IPv6 packets.

- **optname**

Specifies an option. Valid values for this parameter are:

SO_BROADCAST	SO_LINGER	SO_RCVBUF
IP_HDRINCL	IP_TTL	IP_TOS
IP_MULTICAST_TTL	IP_MULTICAST_IF	IP_BROADCAST_IF
IP_RECVIFADDR	TCP_NODELAY	SO_KEEPALIVE
SO_REUSEADDR	IPV6_CHECKSUM	IPV6_DSTOPTS
IPV6_HOPOPTS	IPV6_MULTICAST_HOPS	IPV6_MULTICAST_IF
IPV6_NEXTHOP	IPV6_PKTINFO	IPV6_RECVDSTOPTS
IPV6_RECVHOPLIMIT	IPV6_RECVHOPOPTS	IPV6_RECVPKTINFO
IPV6_RECVRTHDR	IPV6_RECVRTCLASS	IPV6_RTHDR
IPV6_RTHDRDSTOPTS	IPV6_TCLASS	IPV6_UNICAST_HOPS
IPV6_V6ONLY	TCP_FIRST_PROBE_TIMEOUT	TCP_PROBE_TIMEOUT
TCP_MAX_PROBES	TCP_MSL	TCP_FIRST_RTO
TCP_MAX_RTO	TCP_MAX_R2	TCP_MAX_SYN_R2
TCP_DELAY_ACK	TCP_KEEPALIVE_WAIT	TCP_KEEPALIVE_R2
TCP_CONGESTION_CTRL	TCP_CFG_SACK	TCP_WINDOWSCALE
TCP_RCV_WINDOWSIZE	TCP_SND_WINDOWSIZE	TCP_TIMESTAMP

- **optval**

Pointer to the location where the option status can be written.

- **optlen**

Upon calling the function, **optlen** is set to the size of the buffer pointed to by **optval** and modified on return to contain the size of the value returned.

## Return Values

- **NU\_SUCCESS**

Upon successful completion **optval** will contain the status of the option specified by **optname**, and **optlen** will contain the size of the data that was placed into the location pointed to by **optval**. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- **NU\_INVALID\_PARM**  
**optval** was not valid.
- **NU\_INVALID\_SOCKET**

The specified socket descriptor is invalid.

- NU\_INVALID\_LEVEL

The value specified in the level parameter is invalid.

- NU\_INVALID\_OPTION

The value specified in the optname parameter is invalid.

- NU\_NOT\_CONNECTED

A connection does not exist on the specified socket.

### Example

```
INT socketd; /* the socket descriptor */
STATUS status;
INT optstatus = 0;
INT optlen;
.
.
.
optlen = sizeof(optstatus);

/* Retrieve the status of the BROADCAST socket option. */
status = NU_Getsockopt(socketd, SOL_SOCKET, SO_BROADCAST, &optstatus,
                      &optlen);

/* If the call was successful and broadcasting is disabled then
   enable broadcasting. */
if ( (status == NU_SUCCESS) && !(optstatus & SO_BROADCAST) )
{
    optstatus = 1;
    NU_Setsockopt(socketd, SOL_SOCKET, SO_BROADCAST, &optstatus,
                  sizeof(INT));
}

/* Send the datagram. */
bytes_sent = NU_Send_To(socketd, buffer, strlen(buffer), 0, broadaddr,
                        servlen);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>	<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_RCVBUF (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>

## NU\_Getsockopt\_IP\_BROADCAST\_IF (IPv4)

This function gets the broadcast interface of a socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_IP` and type of `IP_BROADCAST_IF`.

### Usage

```
STATUS NU_Getsockopt_IP_BROADCAST_IF (INT    socketd,  
                                     UINT8  *dev_address,  
                                     INT     *addr_len);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `dev_address`  
A pointer to the IP address of the interface used to transmit broadcast packets over the socket.
- `addr_len`  
A pointer to the length of the IP address returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;                /* socket descriptor */  
UINT8  dev_address[4];  
INT     addr_len;  
  
/* Build a socket.*/  
. . .  
  
/* Get the broadcast interface */  
if (NU_Getsockopt_IP_BROADCAST_IF(socketd, dev_address,
```

```
    &addr_len) != NU_SUCCESS)
{
    printf("Could not get broadcast interface().\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Getsockopt\\_SO\\_BROADCAST \(IPv4\)](#)

[NU\\_Setsockopt\\_IP\\_BROADCAST\\_IF \(IPv4\)](#) [NU\\_Setsockopt\\_SO\\_BROADCAST \(IPv4\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Getsockopt\_IP\_HDRINCL (IPv4)

This function determines whether the IP layer will create an IPv4 header for IPRAW packets or if the application layer is responsible for creating the IPv4 header. It performs the same functionality as a call to the function NU\_Getsockopt with a level of IPPROTO\_IP and type of IP\_HDRINCL.

### Usage

```
STATUS NU_Getsockopt_IP_HDRINCL (INT    socketd,  
                                INT16  *optval,  
                                INT    *optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **optval**  
A pointer to the option value. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.
  - **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;                /* socket descriptor */  
INT16  optval;  
INT    optlen;  
  
/* Build an IPRAW socket.*/  
  
/* Determine if the application or IP layer will build the IP
```

```

    * header.
    */
    if (NU_Getsockopt_IP_HDRINCL(socketd, &optval, &optlen) != NU_SUCCESS)
    {
        printf("Could not get IP_HDRINCL option.\n");
    }

```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_IP\\_HDRINC \(IPv4\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Getsockopt\_IP\_MULTICAST\_IF (IPv4)

This function gets the interface to use for sending multicast packets on the socket. It performs the same functionality as a call to the function NU\_Getsockopt with a level of IPPROTO\_IP and type of IP\_MULTICAST\_IF.

### Usage

```
STATUS NU_Getsockopt_IP_MULTICAST_IF (INT      socketd,  
                                     UINT8   *dev_address,  
                                     INT      *addr_len);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **dev\_address**  
A pointer to the IP address of the interface.
- **addr\_len**  
A pointer to the length of the IP address of the interface.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.
  - **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.

### Example

```
INT      socketd;                                /* socket descriptor */  
UINT8   dev_address[4];  
INT      addr_len;  
  
/* Build a socket.*/  
  
/* Get the multicast interface of the socket */  
if (NU_Getsockopt_IP_MULTICAST_IF(socketd, dev_address,  
    &addr_len) != NU_SUCCESS)
```

```
{
    printf("Could not get IP_MULTICAST_IF option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_IP\_MULTICAST\_TTL (IPv4)

This function gets the Time To Live placed in outgoing multicast packets transmitted over the socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_IP` and type of `IP_MULTICAST_TTL`.

### Usage

```
STATUS NU_Getsockopt_IP_MULTICAST_TTL (INT    socketd,  
                                       UINT8  *optval,  
                                       INT     *optlen);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A pointer to the value of the multicast TTL.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `optval` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;                /* socket descriptor */  
UINT8  optval;  
INT     optlen;  
  
/* Build a socket.*/  
  
/* Get the multicast TTL of the socket */  
if (NU_Getsockopt_IP_MULTICAST_TTL(socketd, &optval, &optlen) !=  
    NU_SUCCESS)
```



```
{
    printf("Could not get IP_MULTICAST_TTL option.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Getsockopt\\_IP\\_BROADCAST\\_IF \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_HDRINCL \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_MULTICAST\\_IF \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_RECVIFADDR \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_TOS \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_TTL \(IPv4\)](#)

[NU\\_Getsockopt\\_SO\\_BROADCAST \(IPv4\)](#)

[NU\\_Getsockopt\\_SO\\_KEEPALIVE \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_SO\\_LINGER \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_SO\\_REUSEADDR \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_TCP\\_NODELAY \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_UDP\\_NOCHECKSUM \(IPv4\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Getsockopt\_IP\_RECVIFADDR (IPv4)

This function gets the value of the receive interface address option. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_IP` and type of `IP_RECVIFADDR`.

### Usage

```
STATUS NU_Getsockopt_IP_RECVIFADDR (INT    socketd,  
                                   INT16  *optval,  
                                   INT    *optlen);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A pointer to the value of the receive interface option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `optval` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;                /* socket descriptor */  
INT16  optval;  
INT    optlen;  
  
/* Build a socket.*/  
  
/* Get the value of the receive interface address option */  
if (NU_Getsockopt_IP_RECVIFADDR(socketd, &optval, &optlen) !=
```

```

        NU_SUCCESS)
    {
        printf("Could not get IP_RECVIFADDR option.\n");
    }

```

## Related Topics

[Net API Functions](#)

[NU\\_Getsockopt\\_IP\\_BROADCAST\\_IF \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_HDRINCL \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_MULTICAST\\_IF \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_MULTICAST\\_TTL \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_TOS \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_TTL \(IPv4\)](#)

[NU\\_Getsockopt\\_SO\\_BROADCAST \(IPv4\)](#)

[NU\\_Getsockopt\\_SO\\_KEEPALIVE \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_SO\\_LINGER \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_SO\\_REUSEADDR \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_TCP\\_NODELAY \(IPv4/IPv6\)](#)

[NU\\_Getsockopt\\_UDP\\_NOCHECKSUM \(IPv4\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Getsockopt\_IP\_TOS (IPv4)

This function gets the value of the Type of Service/Differentiated Services Code Point that is being put in outgoing IPv4 packets being transmitted using the socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_IP` and type of `IP_TOS`.

### Usage

```
STATUS NU_Getsockopt_IP_TOS (INT  socketd,  
                             UINT8 *optval,  
                             INT   *optlen);
```

### Arguments

- `socketd`  
Specifies an IPv4 socket descriptor.
- `optval`  
A pointer to the IPv4 TOS/DSCP.
- `optlen`  
A pointer to the length of the value of the option.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.

### Example

```
INT  socketd; /* socket descriptor */  
UINT8 optval;  
INT   optlen;  
  
/* Build a socket.*/  
. . .  
  
/* Get the value of the IP TOS */  
if (NU_Getsockopt_IP_TOS(socketd, &optval, &optlen) != NU_SUCCESS)  
{  
    printf("Could not get IP_TOS option.\n");  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_IP\_TTL (IPv4)

This function gets the value of the Time To Live that is being put in outgoing IP packets that are being transmitted using the socket. It performs the same functionality as a call to the function NU\_Getsockopt with a level of IPPROTO\_IP and type of IP\_TTL.

### Usage

```
STATUS NU_Getsockopt_IP_TTL (INT      socketd,  
                             UINT16  *optval,  
                             INT      *optlen);
```

### Arguments

- **socketd**  
Specifies an IPv4 socket descriptor.
- **optval**  
A pointer to the IP Time to Live.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.
  - **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.

### Example

```
INT      socketd; /* socket descriptor */  
UINT16  optval;  
INT      optlen;  
  
/* Build a socket.*/  
  
/* Get the value of the IP TTL */  
if (NU_Getsockopt_IP_TTL(socketd, &optval, &optlen) !=  
    NU_SUCCESS)
```

```
{
    printf("Could not get IP_TTL option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_CHECKSUM (IPv6)

This function retrieves the value of the socket option for enabling or disabling the socket option for the IP layer to compute the checksum for outgoing IPv6 RAW packets and verifies the checksum for incoming IPv6 RAW packets on the socket.

### Usage

```
STATUS NU_Getsockopt_IPV6_CHECKSUM (INT socketd,  
                                   INT *optval,  
                                   INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
A pointer to the integer offset into the user data of where the checksum is located if this socket option was previously enabled. If this option was not enabled, optval will be -1 upon completion of the function call.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVAL**  
One of the incoming parameters is invalid.

### Example

```
INT socket;  
INT optval;  
INT optlen = sizeof(INT);  
  
/* Create a RAW socket */  
. . .  
  
/* Set the integer offset into the user data of where the  
 * checksum is located.  
 */  
. . .
```



```

/* Get the integer offset into the user data of where the
 * checksum is located.
 */
NU_Getsockopt_IPV6_CHECKSUM(socket, &optval, &optlen);

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_DSTOPTS (IPv6)

This function retrieves the value of the Destination Options Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_DSTOPTS (INT          socketd,  
                                   struct ip6_dest *dst_hdr,  
                                   INT          *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **dst\_hdr**  
A pointer to the Destination Options Sticky Option previously set by the application.
- **optlen**  
Upon calling the function, **optlen** is set to the size of the buffer pointed to by **dst\_hdr** and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT      socket;  
struct ip6_dest dst_hdr  
INT      optlen = sizeof(struct ip6_dest);  
  
/* Get the Destination Options Sticky Option previously set on  
 * this socket.  
 */  
NU_Getsockopt_IPV6_DSTOPTS(socket, &dst_hdr, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_HOPOPTS (IPv6)

This function retrieves the value of the Hop-By-Hop Options Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_HOPOPTS (INT          socketd,  
                                   struct ip6_hbh *hbh_hdr,  
                                   INT          *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **hbh\_hdr**  
A pointer to the Hop-By-Hop Options Sticky Option previously set by the application. The data structure [ip6\\_hbh](#) is described in the [Net Data Structures](#) section.
- **optlen**  
Upon calling the function, **optlen** is set to the size of the buffer pointed to by **hbh\_hdr** and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT      socket;  
struct ip6_hbh hbh_hdr  
INT      optlen = sizeof(struct ip6_hbh);  
  
/* Get the Hop-By-Hop Options Sticky Option previously set on  
 * this socket.  
 */  
NU_Getsockopt_IPV6_HOPOPTS(socket, &hbh_hdr, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_MULTICAST\_HOPS (IPv6)

This function retrieves the value of the hop limit used for outgoing IPv6 multicast packets on the socket.

### Usage

```
STATUS NU_Getsockopt_IPV6_MULTICAST_HOPS (INT socketd,  
                                           INT *hop_limit);
```

### Arguments

- **socketd**  
The socket for which to get the multicast hop limit.
- **hop\_limit**  
A pointer to the hop limit used in multicast packets transmitted on this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
The pointer to hop\_limit is NULL.

### Example

```
INT  socket;  
INT  hop_limit;  
  
/* Get the value of the hop limit used for outgoing multicast  
 * packets on this socket.  
 */  
NU_Getsockopt_IPV6_MULTICAST_HOPS(socket, &hop_limit);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_MULTICAST\_IF (IPv6)

This function retrieves the value of the interface used for outgoing IPv6 multicast packets on the socket.

### Usage

```
STATUS NU_Getsockopt_IPV6_MULTICAST_IF (INT    socketd,  
                                         INT32  *if_index);
```

### Arguments

- `socketd`  
The socket for which to get the option.
- `if_index`  
A pointer to the interface index of the interface used to transmit multicast packets on this socket.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The specified socket descriptor is invalid.
  - `NU_INVALID`  
The pointer to `if_index` is NULL.

### Example

```
INT    socket;  
INT32  if_index;  
  
/* Get the value of the interface used for outgoing multicast  
 * packets on this socket.  
 */  
NU_Getsockopt_IPV6_MULTICAST_IF(socket, &if_index);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_NEXTHOP (IPv6)

This function retrieves the value of the Next-Hop Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_NEXTHOP (INT          socketd,  
                                   struct addr_struct *address,  
                                   INT          *optlen);
```

### Arguments

- `socketd`  
The socket for which to set the option.
- `address`  
A pointer to the [addr\\_struct](#) structure to fill in with the Next-Hop Sticky Option.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `address` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The specified socket descriptor is invalid.
  - `NU_INVALID`  
The pointer to the [addr\\_struct](#) structure is NULL.

### Example

```
INT      socket;  
struct addr_struct  address;  
INT      optlen = sizeof(struct addr_struct);  
  
/* Get the value of the Next-Hop Sticky Option previously set on  
 * this socket.  
 */  
NU_Getsockopt_IPV6_NEXTHOP(socket, &address, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_PKTINFO (IPv6)

This function retrieves the value of the source address and outgoing interface index Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_PKTINFO (INT          socketd,  
                                   in6_pktinfo *pkt_info);
```

### Arguments

- socketd

The socket for which to get the option.

- pkt\_info

A pointer to the [in6\\_pktinfo](#) structure containing the source address and outgoing interface index Sticky Option that was previously set on the socket. If the source address Sticky Option was not set on the socket, the source address will be set to IP6\_ADDR\_ANY. If the interface index Sticky Option was not set, the interface index will be set to IP6\_UNSPECIFIED. Note that this does diverge from RFC 3542 which states a value of 0 should be returned. However, 0 is a valid interface index for Nucleus NET.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- NU\_INVALID\_SOCKET

The specified socket descriptor is invalid.

- NU\_INVALID

The pointer to the [in6\\_pktinfo](#) structure is NULL.

### Example

```
INT          socket;  
in6_pktinfo pkt_info;  
  
/* Get the value of the source address and outgoing interface  
 * index Sticky Option previously set on this socket.  
 */  
NU_Getsockopt_IPV6_PKTINFO(socket, &pkt_info);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVDSTOPTS (IPv6)

This function returns the value of the socket option to receive any Destination Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVDSTOPTS (INT socketd,  
                                       INT *optval,  
                                       INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
A pointer to the value of the option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    opt_val;  
INT    opt_len = sizeof(INT);  
  
/* Determine whether the stack will return the Destination  
 * Options of the most recently received packet for this socket  
 * with the next call to NU_Recvmsg().  
 */  
NU_Getsockopt_IPV6_RECVDSTOPTS(socket, &opt_val, &opt_len);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVHOPLIMIT (IPv6)

This function returns the value of the socket option to receive the Hop Limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVHOPLIMIT (INT socketd,  
                                         INT *optval,  
                                         INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
A pointer to the value of the option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    opt_val;  
INT    opt_len = sizeof(INT);  
  
/* Determine whether the stack will return the Hop Limit of the  
 * most recently received packet for this socket with the next  
 * call to NU_Recvmsg().  
 */  
NU_Getsockopt_IPV6_RECVHOPLIMIT(socket, &opt_val, &opt_len);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVHOPOPTS (IPv6)

This function returns the value of the socket option to receive any Hop-By-Hop Options for the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVHOPOPTS (INT socketd,  
                                       INT *optval,  
                                       INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
A pointer to the value of the option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    opt_val;  
INT    opt_len = sizeof(INT);  
  
/* Determine whether the stack will return the Hop-By-Hop  
 * Options of the most recently received packet for this socket  
 * with the next call to NU_Recvmsg().  
 */  
NU_Getsockopt_IPV6_RECVHOPOPTS(socket, &opt_val, &opt_len);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVPKTINFO (IPv6)

This function returns the value of the socket option to receive the receive interface and destination address of the most recently received IPv6 packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVPKTINFO (INT socketd,  
                                       INT *optval,  
                                       INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
A pointer to the value of the option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    opt_val;  
INT    opt_len = sizeof(INT);  
  
/* Determine whether the stack will return the receive interface  
 * and destination address of the most recently received packet  
 * for this socket with the next call to NU_Recvmsg().  
 */  
NU_Getsockopt_IPV6_RECVPKTINFO(socket, &opt_val, &opt_len);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVRTHDR (IPv6)

This function returns the value of the socket option to receive the Routing Header for the most recently received IPv6 packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVRTHDR (INT  socketd,  
                                     INT  *optval,  
                                     INT  *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
A pointer to the value of the option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT  socket;  
INT  opt_val;  
INT  opt_len = sizeof(INT);  
  
/* Determine whether the stack will return the Routing Header of  
 * the most recently received packet for this socket with the  
 * next call to NU_Recvmsg().  
 */  
NU_Getsockopt_IPV6_RECVRTHDR(socket, &opt_val, &opt_len);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RECVTCLASS (IPv6)

This function determines whether the IPV6\_RECVTCLASS socket option has been set. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Getsockopt_IPV6_RECVTCLASS (INT socketd,  
                                     INT *optval,  
                                     INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
Pointer to the value of the socket option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    optval;  
INT    optlen = sizeof(INT);  
  
/* Determine if the socket option to return the Traffic Class  
 * of the most recently received packet was previously  
 * set on this socket.  
 */  
NU_Getsockopt_IPV6_RECVTCLASS(socket, &optval, &optlen);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RTHDR (IPv6)

This function retrieves the value of the Routing Header Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_RTHDR (INT          socketd,  
                                struct ip6_rthdr *rt_hdr,  
                                INT          *optlen);
```

### Arguments

- `socketd`  
The socket for which to get the option.
- `rt_hdr`  
A pointer to the Routing Header Sticky Option previously set by the application. The [ip6\\_rthdr](#) data structure is defined in the [Net Data Structures](#) section.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `rt_hdr` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The specified socket descriptor is invalid.
  - `NU_INVALID`  
One of the incoming parameters is invalid.

### Example

```
INT      socket;  
struct ip6_rthdr rt_hdr  
INT      optlen = sizeof(struct ip6_rthdr);  
  
/* Get the Routing Header Sticky Option previously set on this  
 * socket.  
 */  
NU_Getsockopt_IPV6_RTHDR(socket, &rt_hdr, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_RTHDRDSTOPTS (IPv6)

This function retrieves the value of the Destination Options Sticky Option for Destination Options that precede the Routing Header that were previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_RTHDRDSTOPTS (INT          socketd,  
                                         struct ip6_dest *dst_hdr,  
                                         INT          *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **dst\_hdr**  
A pointer to the Destination Options Sticky Option previously set by the application.
- **optlen**  
Upon calling the function, **optlen** is set to the size of the buffer pointed to by **dst\_hdr** and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT      socket;  
struct ip6_dest dst_hdr  
INT      optlen = sizeof(struct ip6_dest);  
  
/* Get the Destination Options Sticky Option for Destination  
 * Options that precede the Routing Header previously set on  
 * this socket.  
 */  
NU_Getsockopt_IPV6_RTHDRDSTOPTS(socket, &dst_hdr, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_TCLASS (IPv6)

This function retrieves the value of the Traffic Class Sticky Option that was previously set by the application.

### Usage

```
STATUS NU_Getsockopt_IPV6_TCLASS (INT socketd,  
                                  INT *t_class,  
                                  INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **t\_class**  
A pointer to the Traffic Class Sticky Option previously set by the application. If no Sticky Option value was previously set, this will be the kernel default value as specified by the macro `IP6_TCLASS_DEFAULT` located at *networking/net6\_cfg.h*.
- **optlen**  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `t_class` and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID\_PARM**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    tclass;  
INT    optlen = sizeof(INT);  
  
/* Get the Traffic Class Sticky Option previously set on this  
 * socket.  
 */  
NU_Getsockopt_IPV6_TCLASS(socket, &tclass, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_UNICAST\_HOPS (IPv6)

This function retrieves the value of the Hop Limit for the specified socket.

### Usage

```
STATUS NU_Getsockopt_IPV6_UNICAST_HOPS (INT    socketd,  
                                         INT16 *hop_limit);
```

### Arguments

- **socketd**  
The socket for which to get the Hop Limit.
- **hop\_limit**  
A pointer to the Hop Limit of the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID\_PARM**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    hop_limit;  
  
/* Get the Hop Limit of this socket. */  
NU_Getsockopt_IPV6_UNICAST_HOPS(socket, &hop_limit);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_IPV6\_V6ONLY (IPv6)

This function retrieves the value of the socket option to receive IPv6 packets only on a socket of family type NU\_FAMILY\_IP6.

### Usage

```
STATUS NU_Getsockopt_IPV6_V6ONLY (INT socketd,  
                                  INT *optval,  
                                  INT *optlen);
```

### Arguments

- **socketd**  
The socket for which to get the option.
- **optval**  
Pointer to the value of the option on the socket. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT    optval;  
INT    optlen = sizeof(INT);  
  
/* Determine whether the IPV6_V6ONLY option has been set on the  
 * socket.  
 */  
NU_Getsockopt_IPV6_V6ONLY(socket, &optval, &optlen);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Getsockopt\_SO\_BROADCAST (IPv4)

This function gets the broadcast status of a socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `SOL_SOCKET` and type of `SO_BROADCAST`.

### Usage

```
STATUS NU_Getsockopt_SO_BROADCAST (INT    socketd,  
                                   INT16  *optval,  
                                   INT    *optlen);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A pointer to the value of the broadcast status of the socket. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `optval` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;                /* socket descriptor */  
INT16  optval;  
INT    optlen;  
  
/* Build a socket.*/  
  
/* Determine whether broadcasts are enabled on the socket */  
if (NU_Getsockopt_SO_BROADCAST(socketd, &optval, &optlen) !=  
    NU_SUCCESS)
```

```
{
    printf("Could not get SO_BROADCAST option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_SO\_KEEPALIVE (IPv4/IPv6)

This function determines whether TCP Keep-Alive has been enabled on a socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_TCP` and type of `SO_KEEPALIVE`. By default, Keep-Alive is disabled on all sockets.

### Usage

```
STATUS NU_Getsockopt_SO_KEEPALIVE (INT socketd,  
                                   INT *optval,  
                                   INT *optlen);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A pointer to the option value. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `optval` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_INVAL`  
There is no port structure associated with the socket.
  - `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.

### Example

```
INT    socketd;           /* socket descriptor */  
INT    optval;  
INT    optlen;
```

```

/* Build a TCP socket.*/

/* Determine if TCP Keep-Alive is enabled on this socket */
if (NU_Getsockopt_SO_KEEPALIVE(socketd, &optval, &optlen) !=
    NU_SUCCESS)
{
    printf("Could not get SO_KEEPALIVE option.\n");
}

```

## Related Topics

Net API Functions	NU_Getsockopt_IP_BROADCAST_IF (IPv4)
NU_Getsockopt_IP_HDRINCL (IPv4)	NU_Getsockopt_IP_MULTICAST_IF (IPv4)
NU_Getsockopt_IP_MULTICAST_TTL (IPv4)	NU_Getsockopt_IP_RECVIFADDR (IPv4)
NU_Getsockopt_IP_TOS (IPv4)	NU_Getsockopt_IP_TTL (IPv4)
NU_Getsockopt_IPV6_CHECKSUM (IPv6)	NU_Getsockopt_IPV6_DSTOPTS (IPv6)
NU_Getsockopt_IPV6_HOPOPTS (IPv6)	NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)
NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)	NU_Getsockopt_IPV6_NEXTHOP (IPv6)
NU_Getsockopt_IPV6_PKTINFO (IPv6)	NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)
NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)	NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)
NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)	NU_Getsockopt_IPV6_RECVRTHDR (IPv6)
NU_Getsockopt_IPV6_RECVTCLASS (IPv6)	NU_Getsockopt_IPV6_RTHDR (IPv6)
NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)	NU_Getsockopt_IPV6_TCLASS (IPv6)
NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)	NU_Getsockopt_IPV6_V6ONLY (IPv6)
NU_Getsockopt_SO_BROADCAST (IPv4)	NU_Getsockopt_SO_LINGER (IPv4/IPv6)
NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)	NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)
NU_Getsockopt_UDP_NOCHECKSUM (IPv4)	NU_Getsockopt (IPv4/IPv6)

## NU\_Getsockopt\_SO\_LINGER (IPv4/IPv6)

This function gets the settings for the SO\_LINGER option. It performs the same functionality as a call to the function NU\_Getsockopt with a level of SOL\_SOCKET and type of SO\_LINGER.

### Usage

```
STATUS NU_Getsockopt_SO_LINGER (INT socketd,
                                struct sck_linger_struct *optval,
                                INT optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **optval**  
A pointer to the [sck\\_linger\\_struct](#) structure in which to store the linger settings. A non-zero value in the s\_linger.linger\_on parameter indicates that the SO\_LINGER option is enabled on the socket, and the s\_linger.linger\_ticks parameter contains the number of ticks the socket will linger before closing.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.

### Example

```
INT  socketd;    /* socket descriptor */
struct sck_linger_struct optval;
INT  optlen;

/* Build a socket.*/

/* Get the linger settings for the socket */
if (NU_Getsockopt_SO_LINGER (socketd, &optval, &optlen) !=
    NU_SUCCESS)
```



```
{
    printf("Could not get SO_LINGER option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_SO\_RCVBUF (IPv4/IPv6)

This function retrieves the TCP Window Size of the local side of the connection for a specified socket. This routine has been deprecated in favor of

[NU\\_Getsockopt\\_TCP\\_RCV\\_WINDOWSIZE](#) in order to take advantage of the expanded Window Size field enabled by using the TCP Window Scale option.

### Usage

```
STATUS NU_Getsockopt_SO_RCVBUF (INT    socketd,  
                                INT32  *opt_val);
```

### Arguments

- **socketd**  
The TCP socket descriptor for which to retrieve the local TCP Window Size.
- **opt\_val**  
Pointer to the local TCP Window Size set on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
  - **NU\_INVALID\_PARM**  
The input parameter `opt_val` is NULL.

### Example

```
INT    socketd;  
struct addr_struct servaddr;  
INT32  opt_val;  
  
/* Create a TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
/* Fill in the addr_struct. */  
. . .  
  
/* Initiate a connection using the TCP socket. */  
if (NU_Connect(socketd, &servaddr, 0) >= 0)  
{  
    /* Get the TCP Window Size for the local side of the connection */  
    status = NU_Getsockopt_SO_RCVBUF(socketd, &opt_val);  
  
    if (status != NU_SUCCESS)
```

```
{  
    printf("Error at call to NU_Getsockopt_SO_RCVBUF.\n");  
}  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_Setsockopt\\_SO\\_RCVBUF \(IPv4/IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Getsockopt\_SO\_REUSEADDR (IPv4/IPv6)

This function gets the settings for the SO\_REUSEADDR option. It performs the same functionality as a call to the function NU\_Getsockopt with a level of SOL\_SOCKET and type of SO\_REUSEADDR.

### Usage

```
STATUS NU_Getsockopt_SO_REUSEADDR (INT    socketd,  
                                   INT16  *optval,  
                                   INT    *optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **optval**  
A pointer to the memory in which to store the value of the SO\_REUSEADDR option. A value of 1 indicates that the socket option is enabled. A value of 0 indicates that the socket option is not enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.

### Example

```
INT    socketd; /* socket descriptor */  
INT16  optval;  
INT    optlen;  
  
/* Build a socket.*/  
  
/* Get the SO_REUSEADDR settings for the socket */  
if (NU_Getsockopt_SO_REUSEADDR (socketd, &optval, &optlen) !=  
    NU_SUCCESS)
```

```
{
    printf("Could not get SO_REUSEADDR option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_TCP\_CFG\_DSACK

This function determines whether Duplicate Selective Acknowledgement (D-SACK) support is enabled on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_CFG_DSACK(INT    socketd,  
                                   UINT8  *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT8        opt_val;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the D-SACK configuration of the socket. */  
    status = NU_Getsockopt_TCP_CFG_DSACK(socketd, &opt_val);  
}
```

```
    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_CFG\_SACK

This function determines whether Selective Acknowledgement (SACK) support is enabled on a TCP socket at run-time. If you enable SACK on a socket, but the foreign node does not support SACK, this routine will inform you that SACK has been disabled on the socket by the underlying protocol stack.

### Usage

```
STATUS NU_Getsockopt_TCP_CFG_SACK (INT    socketd,  
                                   UINT8  *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option. A zero value indicates that SACK is disabled. A non-zero value indicates that SACK is enabled.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL

### Example

```
INT          socketd;  
UINT8        opt_val  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */
```



```

    . . .

    /* Retrieve the SACK configuration of the socket. */
    status = NU_Getsockopt_TCP_CFG_SACK(socketd, &opt_val);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_CONGESTION\_CTRL

This function determines whether TCP Congestion Control is enabled on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_CONGESTION_CTRL(INT    socketd,  
                                         UINT8  *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT8       opt_val  
STATUS      status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the TCP Congestion Control configuration of the  
     * socket.
```

```

    */
    status = NU_Getsockopt_TCP_CONGESTION_CTRL(socketd, &opt_val);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_DELAY\_ACK

This function retrieves the timeout value used for delaying successive TCP ACKs on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_DELAY_ACK (INT      socketd,  
                                     UINT32  *delay)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **delay**  
Pointer to the memory in which to store the value in hardware ticks to delay transmitting successive ACKs for incoming data for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT      socketd;  
UINT32   delay;  
STATUS    status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the amount of time to delay sending ACKs for
```

```

        * incoming data.
        */
        status = NU_Getsockopt_TCP_DELAY_ACK(socketd, &delay);

        if (status == NU_SUCCESS)
        {
            . . .
        }
    }

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_FIRST\_RTO

This function retrieves the value of the first retransmission timeout on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_FIRST_RTO(INT    socketd,  
                                   INT32  *timeout)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **timeout**  
Pointer to the memory in which to store the value in hardware ticks to delay before sending the first retransmission of an unACKed data packet for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL.

### Example

```
INT          socketd;  
INT32        timeout;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the first retransmission timeout configured  
     * for the socket.
```

```

        */
        status = NU_Getsockopt_TCP_FIRST_RTO(socketd, &timeout);

        if (status == NU_SUCCESS)
        {
            . . .
        }
    }

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_FIRST\_TIMEOUT

This function retrieves the timeout value for sending the initial Zero Window probe on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_FIRST_PROBE_TIMEOUT(INT    socketd,  
                                              INT32  *timeout)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **timeout**  
Pointer to the memory in which to store the value in hardware ticks to delay before sending the initial Zero Window probe on the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
INT32        timeout;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the value for sending the initial Zero Window
```



```

    * probe over the connection.
    */
    status =
        NU_Getsockopt_TCP_FIRST_PROBE_TIMEOUT(socketd, &timeout);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_KEEPALIVE\_R2

This function retrieves the maximum number of unanswered keep-alive probes to transmit on a TCP socket at run-time before closing the connection.

### Usage

```
STATUS NU_Getsockopt_TCP_KEEPALIVE_R2 (INT    socketd,  
                                       UINT8  *max_retrans)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **max\_retrans**  
The pointer to memory in which to store the maximum number of retransmissions of unanswered Keep-Alive packets for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT8        max_retrans;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the total number of Keep-Alive packets to transmit
```

```
    * before closing the connection.
    */
    status = NU_Getsockopt_TCP_KEEPALIVE_R2(socketd, &max_retrans);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_KEEPALIVE\_WAIT

This function retrieves the time value to remain idle on a TCP socket before invoking the TCP keep-alive module at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_KEEPALIVE_WAIT(INT    socketd,  
                                         UINT32 *delay)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **delay**  
The pointer to memory in which to store the value in hardware ticks to allow a connection to remain idle before invoking the keep-alive protocol.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT32      delay;  
STATUS      status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the amount of time to delay before invoking the
```

```
    * Keep-Alive protocol on an idle connection.
    */
    status = NU_Getsockopt_TCP_KEEPALIVE_WAIT(socketd, &delay);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_MAX\_PROBES

This function retrieves the maximum number of Zero Window probes to be transmitted before giving up on window probing on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_MAX_PROBES(INT    socketd,  
                                     UINT8  *max_probes)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **max\_probes**  
Pointer to the memory in which to store the maximum number of unanswered Zero Window probes to transmit over the specified socket before closing the connection.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to **NU\_Socket**.
- **NU\_INVALID\_PARM**  
The memory in which to store the return value is **NULL**.

### Example

```
INT          socketd;  
UINT8        max_probes;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the maximum number of Zero Window probes to be
```

```
    * transmitted before closing the connection.
    */
    status =
        NU_Getsockopt_TCP_MAX_PROBES(socketd, &max_probes);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_MAX\_R2

This function retrieves the number of times to retransmit a data segment before closing the connection on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_MAX_R2 (INT    socketd,  
                                UINT8  *max_retrans)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **max\_retrans**  
Pointer to the memory in which to store the maximum number of retransmissions of an unACKed data packet for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL.

### Example

```
INT          socketd;  
UINT8       max_retrans;  
STATUS      status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the maximum number of retransmissions to be
```



```

    * sent from the socket.
    */
    status = NU_Getsockopt_TCP_MAX_R2(socketd, &max_retrans);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_MAX\_RTO

This function retrieves the value of the maximum retransmission timeout on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_MAX_RTO (INT      socketd,  
                                  UINT32  *timeout)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **timeout**  
Pointer to the memory in which to store the maximum value in hardware ticks to delay between successive retransmissions of an unACKed data packet for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL.

### Example

```
INT      socketd;  
UINT32  timeout;  
STATUS  status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the maximum retransmission timeout configured
```

```
    * for the socket.  
    */  
    status = NU_Getsockopt_TCP_MAX_RTO(socketd, &timeout);  
  
    if (status == NU_SUCCESS)  
    {  
        . . .  
    }  
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_MAX\_SYN\_R2

This function retrieves the number of times to retransmit a SYN packet before closing the connection on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_MAX_SYN_R2 (INT    socketd,  
                                     UINT8  *max_retrans)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **max\_retrans**  
Pointer to the memory in which to store the maximum number of retransmissions of an unACKed SYN packet for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL.

### Example

```
INT          socketd;  
UINT8        max_retrans;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the maximum number of retransmissions of the
```

```

    * outgoing SYN packet to be sent from the socket.
    */
    status = NU_Getsockopt_TCP_MAX_SYN_R2(socketd, &max_retrans);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_MSL

This function retrieves the amount of time to wait before reusing a closed port / IP address combination on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_MSL(INT    socketd,  
                             UINT32 *msl)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **msl**  
Pointer to the memory in which to store the MSL value in hardware ticks for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to **NU\_Socket**.
- **NU\_INVALID\_PARM**  
The memory in which to store the return value is **NULL**.

### Example

```
INT          socketd;  
UINT32       msl;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the MSL configured for the socket. */
```

```
status = NU_Getsockopt_TCP_MSL(socketd, &mssl);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_NODELAY (IPv4/IPv6)

This function gets the value of the Nagle Algorithm for the socket. It performs the same functionality as a call to the function `NU_Getsockopt` with a level of `IPPROTO_TCP` and type of `TCP_NODELAY`.

### Usage

```
STATUS NU_Getsockopt_TCP_NODELAY (INT socketd,  
                                  INT *optval,  
                                  INT *optlen);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A pointer to the value of the Nagle Algorithm. A value of zero indicates that the Nagle Algorithm is enabled; that is, transmission of data is delayed until a MSS size packet is created or `SWSOVERRIDE` seconds have passed since the initial transmission was invoked. A non-zero value indicates that the Nagle Algorithm is disabled; that is, transmission of data is not delayed regardless of packet size.
- `optlen`  
Upon calling the function, `optlen` is set to the size of the buffer pointed to by `optval` and modified on return to contain the size of the value returned.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - `NU_INVALID_PARM`  
One of the parameters provided is invalid.
  - `NU_INVALID`  
There is no port associated with the socket.
  - `NU_NOT_CONNECTED`
    - a. A connection does not exist on the specified socket.



## Example

```
INT    socketd;                /* socket descriptor */
INT    optval;
INT    optlen;

/* Build a TCP socket.*/

/* Get the value of the Nagle Algorithm for the socket */
if (NU_Getsockopt_TCP_NODELAY(socketd, &optval, &optlen) !=
    NU_SUCCESS)
{
    printf("Could not get TCP_NODELAY option.\n");
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Getsockopt\_TCP\_PROBE\_TIMEOUT

This function retrieves the maximum timeout value for sending successive Zero Window probes on a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_PROBE_TIMEOUT(INT    socketd,  
                                       INT32  *timeout)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **timeout**  
Pointer to the memory in which to store the value in hardware ticks to delay between successive Zero Window probes on the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The memory in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
INT32        timeout;  
STATUS        status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */  
    . . .  
  
    /* Retrieve the maximum timeout value for sending successive
```

```
    * Zero Window probes over the connection.
    */
    status =
        NU_Getsockopt_TCP_PROBE_TIMEOUT(socketd, &timeout);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_RCV\_WINDOWSIZE

This function retrieves the local receive window size for a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_RCV_WINDOWSIZE(INT    socketd,  
                                         UINT32 *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT32       opt_val;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Retrieve the local window size for the socket. */  
    status = NU_Getsockopt_TCP_RCV_WINDOWSIZE(socketd, &opt_val);  
  
    if (status == NU_SUCCESS)  
    {  
        . . .  
    }  
}
```

NET  
NU\_Getsockopt\_TCP\_RCV\_WINDOWSIZE

---

```
    }  
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_SND\_WINDOWSIZE

This function retrieves the foreign receive window size for a TCP socket at run-time.

### Usage

```
STATUS NU_Getsockopt_TCP_SND_WINDOWSIZE(INT    socketd,  
                                         UINT32 *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT32      opt_val;  
STATUS      status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Retrieve the foreign window size for the socket. */  
    status = NU_Getsockopt_TCP_SND_WINDOWSIZE(socketd, &opt_val);  
  
    if (status == NU_SUCCESS)  
    {  
        . . .  
    }  
}
```

NET  
NU\_Getsockopt\_TCP\_SND\_WINDOWSIZE

---

```
    }  
}
```

## Related Topics

[Net API Functions](#)



## NU\_Getsockopt\_TCP\_TIMESTAMP

This function determines whether the TCP Timestamp option is enabled on a socket at run-time. If you enable the Timestamp option on a socket, but the foreign node does not support this option, this routine will inform you that the Timestamp option has been disabled on the socket by the underlying protocol stack.

### Usage

```
STATUS NU_Getsockopt_TCP_TIMESTAMP(INT    socketd,  
                                   UINT8  *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is `NULL`.

### Example

```
INT          socketd;  
UINT8       opt_val;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Determine whether the Timestamp option is enabled. */  
    status = NU_Getsockopt_TCP_TIMESTAMP(socketd, &opt_val);  
}
```

```
        if (status == NU_SUCCESS)
        {
            . . .
        }
    }
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_TCP\_WINDOWSCALE

This function determines whether support for the TCP Window Scale option is enabled on a TCP socket at run-time. If support is enabled, the routine returns the value of the shift count for the socket.

### Usage

```
STATUS NU_Getsockopt_TCP_WINDOWSCALE (INT socketd,  
                                       INT *optval)
```

### Arguments

- **socketd**  
Socket for which to retrieve the value.
- **optval**  
Pointer to the memory in which to store the value of the option. If the option is disabled, this value will be set to -1. If the option is enabled, this value will be set to the configured value of the shift count for the TCP Window Scale option on the socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVALID\_PARM**  
The pointer in which to store the return value is NULL.

### Example

```
INT          socketd;  
INT          opt_val;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket. */
```

```
    . . .

    /* Retrieve the Window Scale configuration of the socket. */
    status = NU_Getsockopt_TCP_WINDOWSCALE(socketd, &opt_val);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

## NU\_Getsockopt\_UDP\_NOCHECKSUM (IPv4)

This routine determines whether the UDP\_NOCHECKSUM option is enabled on the IPv4 socket.

### Usage

```
STATUS NU_Getsockopt_UDP_NOCHECKSUM (INT    socketd,
                                     INT16  *optval,
                                     INT    *optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **optval**  
A pointer to the value of the UDP\_NOCHECKSUM option. A value of zero indicates that the socket option is disabled. A non-zero value indicates that the socket option is enabled.
- **optlen**  
Upon calling the function, optlen is set to the size of the buffer pointed to by optval and modified on return to contain the size of the value returned.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.
  - **NU\_NOT\_CONNECTED**  
The socket has been closed.

### Example

```
INT    socketd;           /* socket descriptor */
INT16  optval;
INT    optlen;

/* Build a UDP socket.*/

/* Determine whether the UDP_NOCHECKSUM option is enabled. */
if (NU_Getsockopt_UDP_NOCHECKSUM(socketd, &optval, &optlen) !=
    NU_SUCCESS)
```

```
{  
    printf("Could not get UDP_NOCHECKSUM option.\n");  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Getsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_HDRINCL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Getsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Getsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Getsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Getsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Getsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Getsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Getsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Getsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Getsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_IF\_FreeNameIndex (IPv4/IPv6)

This function frees the dynamic memory that was allocated by [NU\\_IF\\_NameIndex\(\)](#).

### Usage

```
VOID NU_IF_FreeNameIndex (struct if_nameindex *ptr);
```

### Arguments

- ptr  
A pointer to the memory returned by [NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#). Structure [if\\_nameindex](#) is defined in the [Net Data Structures](#) section.

### Example

```
struct if_nameindex *ni_ptr, *tmp_ptr;

/* Get a list of interfaces. */
ni_ptr = NU_IF_NameIndex();

tmp_ptr = ni_ptr;

/* Traverse the list. */
while ( (ni_ptr->if_index) || (ni_ptr->if_name) )
{
    ni_ptr = (struct if_nameindex *)
        ((CHAR HUGE*)ni_ptr + sizeof(struct if_nameindex) +
        DEV_NAME_LENGTH);

    . . .
}

/* Free the memory. */
NU_IF_FreeNameIndex(tmp_ptr);
```

### Related Topics

[Net API Functions](#)

[NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_NameToIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_IndexToName \(IPv4/IPv6\)](#)

## NU\_IF\_IndexToName (IPv4/IPv6)

This function maps an interface index to its corresponding name.

### Usage

```
CHAR *NU_IF_IndexToName (INT32 if_index,  
                          CHAR *if_name);
```

### Arguments

- `if_index`  
The interface index of an interface in the system.
- `if_name`  
When the function is called, this argument must point to a buffer of at least `DEV_NAME_LENGTH` bytes. The function places in this buffer the name of the interface with index `if_index`.

### Return Values

- The interface name corresponding to the interface index.  
Upon successful completion.
- `NU_NULL`  
Otherwise.

### Example

```
CHAR    if_name[DEV_NAME_LENGTH];  
INT32    if_index = 1;  
  
if (NU_IF_IndexToName(if_index, if_name) != NU_NULL)  
    . . .
```

### Related Topics

[Net API Functions](#)

[NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_NameToIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_FreeNameIndex \(IPv4/IPv6\)](#)



## NU\_IF\_NameIndex (IPv4/IPv6)

This function returns an array of all interface names and indexes on the node. The memory used for the array is obtained dynamically and must be freed by `NU_IF_FreeNameIndex()`.

### Usage

```
struct if_nameindex *NU_IF_NameIndex (VOID);
```

### Return Values

- An array of [if\\_nameindex](#) structures, one structure per interface. The end of the array of structures is indicated by a structure with an `if_index` of 0 and an `if_name` of `NU_NULL`.

Upon successful completion.

- -1

Otherwise.

### Example

```
struct if_nameindex *ni_ptr, *tmp_ptr;

/* Get a list of interfaces. */
ni_ptr = NU_IF_NameIndex();

tmp_ptr = ni_ptr;

/* Traverse the list. */
while ( (ni_ptr->if_index) || (ni_ptr->if_name) )
{
    ni_ptr = (struct if_nameindex *)
        ((CHAR HUGE*)ni_ptr + sizeof(struct if_nameindex) +
        DEV_NAME_LENGTH);

    . . .
}

/* Free the memory. */
NU_IF_FreeNameIndex(tmp_ptr);
```

### Related Topics

[Net API Functions](#)

[NU\\_IF\\_IndexToName \(IPv4/IPv6\)](#)

[NU\\_IF\\_NameToIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_FreeNameIndex \(IPv4/IPv6\)](#)

## NU\_IF\_NameToIndex (IPv4/IPv6)

This function maps an interface name to its corresponding index.

### Usage

```
INT32 NU_IF_NameToIndex (CHAR *if_name);
```

### Arguments

- `if_name`  
Pointer to the name of an interface in the system.

### Return Values

- The interface index corresponding to the interface name.  
Upon successful completion.  
Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `-1`  
No interface exists with the specified name.
  - `NU_INVALID_PARM`  
The pointer passed into the routine is NULL.

### Example

```
CHAR    if_name[]  = "eth0_dev";  
INT32   if_index;  
  
if_index = NU_IF_NameToIndex(if_name);  
  
if (if_index >= 0)  
    . . .
```

### Related Topics

[Net API Functions](#)

[NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#)

[NU\\_IF\\_IndexToName \(IPv4/IPv6\)](#)

[NU\\_IF\\_FreeNameIndex \(IPv4/IPv6\)](#)

## NU\_Inet6\_Opt\_Append (IPv6)

This function returns the updated total length, taking into account adding an option with length “len” and alignment “align”.

### Usage

```
INT NU_Inet6_Opt_Append (VOID    *extbuf,  
                        UINT32  extlen,  
                        INT      offset,  
                        UINT8    type,  
                        UINT32  len,  
                        UINT8    align,  
                        VOID     **databufp);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **extlen**  
The length of the buffer extbuf.
- **offset**  
The length returned by NU\_Inet6\_Opt\_Init() or a previous call to NU\_Inet6\_Opt\_Append().
- **type**  
The 8-bit option type. Must have a value from 2 to 255, inclusive. (0 and 1 are reserved for the Pad1 and PadN options, respectively.)
- **len**  
The length of the option data, excluding the option type and option length fields. Must have a value between 0 and 255, inclusive, and is the length of the option data that follows.
- **align**  
Must have a value of 1, 2, 4, or 8. The align value cannot exceed the value of len.
- **databufp**  
A pointer to the location for the option content.

### Return Values

- The updated total length of the buffer.  
Upon successful completion. In addition to returning the length, the function inserts any needed pad option, initializes the option (setting the type and length fields), and returns a pointer to the location for the option content in databufp.
- -1  
If the option does not fit in the extension header buffer.

## Description

If extbuf is not NULL then, in addition to returning the length, the function inserts any needed pad option, initializes the option (setting the type and length fields), and returns a pointer to the location for the option content in databuf. If the option does not fit in the extension header buffer the function returns -1.

Once NU\_Inet6\_Opt\_Append() has been called, the application can use the databuf directly, or use NU\_Inet6\_Opt\_Set\_Val() to specify the content of the option.

## Example

```
INT      socket;
INT      currentlen;
UINT32   extlen;
VOID     *databuf;
INT      offset;
UINT8    value1;
UINT16   value2;
UINT32   value4;
UINT32   value8;
struct ip6_dest *dst_hdr;

/** Build two Destination Options. */

/* Determine the amount of memory for the Destination Options. */
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);

currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X,
                                12, 8, NU_NULL);

currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y,
                                7, 4, NU_NULL);

currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID**) &dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure. */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
```

```

offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof
                               (value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                  OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for a 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof
                               (value1));

/* Insert value 0x1331 for a 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof
                               (value2));

/* Insert value 0x01020304 for a 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                               (value4));

currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Inet6_Opt_Finish (IPv6)</a>
<a href="#">NU_Inet6_Opt_Get_Val (IPv6)</a>	<a href="#">NU_Inet6_Opt_Init (IPv6)</a>
<a href="#">NU_Inet6_Opt_Next (IPv6)</a>	<a href="#">NU_Inet6_Opt_Set_Val (IPv6)</a>
<a href="#">NU_Inet6_Rth_Add (IPv6)</a>	<a href="#">NU_Inet6_Rth_GetAddr (IPv6)</a>
<a href="#">NU_Inet6_Rth_Init (IPv6)</a>	<a href="#">NU_Inet6_Rth_Reverse (IPv6)</a>
<a href="#">NU_Inet6_Rth_Segments (IPv6)</a>	<a href="#">NU_Inet6_Rth_Space (IPv6)</a>
<a href="#">NU_Inet6_Opt_Find (IPv6)</a>	

## NU\_Inet6\_Opt\_Find (IPv6)

This function is similar to NU\_Inet6\_Opt\_Next(), except this function lets the caller specify the option type to be searched for instead of always returning the next option in the extension header.

### Usage

```
INT NU_Inet6_Opt_Find (VOID    *extbuf,  
                      UINT32  extlen,  
                      INT     offset,  
                      UINT8   type,  
                      UINT32  *lenp,  
                      VOID    **databufp);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **extlen**  
The length of the extension header buffer.
- **offset**  
Either zero (for the first option) or the length returned by a previous call to NU\_Inet6\_Opt\_Next() or NU\_Inet6\_Opt\_Find(). It specifies the position where to continue scanning the extension buffer.
- **type**  
The option type.
- **lenp**  
Pointer to the length of the option data (excluding the option type and option length fields).
- **databufp**  
Points to the data field of the option.

### Return Values

- If an option of the specified type is located, the function returns the updated “previous” total length.
- -1  
If an option of the specified type is not located, or if the option extension header is malformed.

### Description

If an option of the specified type is located, the function returns the updated “previous” total length computed by advancing past the option that was returned and past any options that did not match the type. This returned “previous” length can then be passed to subsequent calls to NU\_Inet6\_Opt\_Find() for finding the next occurrence of the same option type.

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Opt\_Finish (IPv6)

This function returns the updated total length taking into account the final padding of the extension header to make it a multiple of 8 bytes.

### Usage

```
INT NU_Inet6_Opt_Finish (VOID    *extbuf,  
                        UINT32  extlen,  
                        INT      offset);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **extlen**  
The length of the buffer extbuf.
- **offset**  
The length returned by NU\_Inet6\_Opt\_Init() or a previous call to NU\_Inet6\_Opt\_Append().

### Return Values

- The updated total length of the buffer.  
Upon successful completion.
- -1  
Otherwise.

### Description

If extbuf is not NU\_NULL, the function also initializes the option by inserting a Pad1 or PadN option of the proper length.

If the necessary pad does not fit in the extension header buffer, the function returns -1.

### Example

```
INT      socket;  
INT      currentlen;  
UINT32   extlen;  
VOID     *databuf;  
INT      offset;  
UINT8    value1;  
UINT16   value2;  
UINT32   value4;  
UINT32   value8;  
struct   ip6_dest  *dst_hdr;  
  
/** Build two Destination Options. */  
  
/* Determine the amount of memory for the Destination Options. */  
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);
```



```
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X,
                                12, 8, NU_NULL);

currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y,
                                7, 4, NU_NULL);

currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID*)&dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof
                              (value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for a 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof
                              (value1));

/* Insert value 0x1331 for a 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof
                              (value2));

/* Insert value 0x01020304 for a 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Opt\_Get\_Val (IPv6)

This function extracts data items of various sizes in the data portion of the option. It is expected that each field is aligned on its natural boundaries as described in “Appendix B” of *RFC 2460*, but the function must not rely on the alignment.

### Usage

```
INT NU_Inet6_Opt_Get_Val (VOID    *databuf,
                          UINT32  offset,
                          VOID    *val,
                          INT      vallen);
```

### Arguments

- **databuf**  
 A pointer returned by [NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#) or [NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#).
- **offset**  
 The point in the data portion of the option the value should be extracted. The first byte after the option type and length is accessed by specifying an offset of zero.
- **val**  
 A pointer to the destination in which to place the extracted data.
- **vallen**  
 The length of the data stored in val.

### Return Values

- The offset for the next field (for example, offset + vallen), which can be used when extracting option content with multiple fields.

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Inet6_Opt_Find (IPv6)</a>
<a href="#">NU_Inet6_Opt_Finish (IPv6)</a>	<a href="#">NU_Inet6_Opt_Init (IPv6)</a>
<a href="#">NU_Inet6_Opt_Next (IPv6)</a>	<a href="#">NU_Inet6_Opt_Set_Val (IPv6)</a>
<a href="#">NU_Inet6_Rth_Add (IPv6)</a>	<a href="#">NU_Inet6_Rth_GetAddr (IPv6)</a>
<a href="#">NU_Inet6_Rth_Init (IPv6)</a>	<a href="#">NU_Inet6_Rth_Reverse (IPv6)</a>
<a href="#">NU_Inet6_Rth_Segments (IPv6)</a>	<a href="#">NU_Inet6_Rth_Space (IPv6)</a>
<a href="#">NU_Inet6_Opt_Append (IPv6)</a>	

## NU\_Inet6\_Opt\_Init (IPv6)

This function returns the number of bytes needed for the empty extension header; for example, without any options.

### Usage

```
INT NU_Inet6_Opt_Init (VOID    *extbuf,  
                      UINT32  extlen);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **extlen**  
The length of the buffer extbuf.

### Return Values

- The number of bytes required for the empty extension header.  
Upon successful completion.
- -1  
Otherwise.

### Description

If extbuf is not NU\_NULL, it also initializes the extension header to have the correct length field. In that case, if the extlen value is not a positive (for example, non-zero) multiple of 8, the function fails and returns -1.

### Example

```
INT      socket;  
INT      currentlen;  
UINT32   extlen;  
VOID     *databuf;  
INT      offset;  
UINT8    value1;  
UINT16   value2;  
UINT32   value4;  
UINT32   value8;  
struct ip6_dest *dst_hdr;  
  
/** Build two Destination Options. */  
  
/* Determine the amount of memory for the Destination Options. */  
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);  
  
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X,  
                                12, 8, NU_NULL);  
  
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y,  
                                7, 4, NU_NULL);
```

```
currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID**)&dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure. */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof
                              (value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for a 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof
                              (value1));

/* Insert value 0x1331 for a 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof
                              (value2));

/* Insert value 0x01020304 for a 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Opt\_Next (IPv6)

This function parses received option extension headers returning the next option.

### Usage

```
INT NU_Inet6_Opt_Next (VOID    *extbuf,  
                      UINT32  extlen,  
                      INT      offset,  
                      UINT8    *typep,  
                      UINT8    *lenp,  
                      VOID     **databufp);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **extlen**  
The length of the extension header buffer.
- **offset**  
Either zero (for the first option) or the length returned by a previous call to NU\_Inet6\_Opt\_Next() or NU\_Inet6\_Opt\_Find(). It specifies the position where to continue scanning the extension buffer.
- **typep**  
Pointer to the option type.
- **lenp**  
Pointer to the length of the option data (excluding the option type and option length fields).
- **databufp**  
Points to the data field of the option.

### Return Values

- The updated previous length.
- -1  
When there are no more options or if the option extension header is malformed.

### Description

The next option is returned by updating typep, lenp, and databufp. This function returns the updated “previous” length computed by advancing past the option that was returned. This returned “previous” length can then be passed to subsequent calls to NU\_Inet6\_Opt\_Next(). This function does not return any PAD1 or PADN options.

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)



## NU\_Inet6\_Opt\_Set\_Val (IPv6)

This function inserts data items of various sizes in the data portion of the option.

### Usage

```
INT NU_Inet6_Opt_Set_Val (VOID    *extbuf,  
                          UINT32  offset,  
                          VOID    *val,  
                          INT      vallen);
```

### Arguments

- **extbuf**  
A pointer to the buffer containing the extension header being built.
- **offset**  
Specifies where in the data portion of the option the value should be inserted; the first byte after the option type and length is accessed by specifying an offset of zero.
- **val**  
A pointer to the data to be inserted.
- **vallen**  
The number of bytes to insert.

### Return Values

- The offset for the next field (for example, offset + vallen), which can be used when composing option content with multiple fields.

### Description

The caller should ensure that each field is aligned on its natural boundaries as described in "Appendix B" of *RFC 2460*, but the function must not rely on the caller's behavior. Even when the alignment requirement is not satisfied, `NU_Inet6_Opt_Set_Val()` should just copy the data as required.

### Example

```
INT      socket;  
INT      currentlen;  
UINT32   extlen;  
VOID     *databuf;  
INT      offset;  
UINT8    value1;  
UINT16   value2;  
UINT32   value4;  
UINT32   value8;  
struct ip6_dest *dst_hdr;  
  
/** Build two Destination Options. */  
  
/* Determine the amount of memory for the Destination Options. */  
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);
```

```
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X,
                                12, 8, NU_NULL);

currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y,
                                7, 4, NU_NULL);

currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID**)&dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure. */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof
                              (value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for an 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof
                              (value1));

/* Insert value 0x1331 for a 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof
                              (value2));

/* Insert value 0x01020304 for a 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_Add (IPv6)

This function adds the IPv6 address pointed to by `addr` to the end of the Routing Header being constructed.

### Usage

```
INT *NU_Inet6_Rth_Add (VOID *bp,  
                      UINT8 *addr);
```

### Arguments

- `bp`  
A pointer to the Routing Header being constructed.
- `addr`  
A pointer to the IPv6 address to add to the Routing Header.

### Return Values

- 0  
Upon successful completion.
- -1  
Otherwise.

### Example

```
UINT32    rt_space;  
  
/** Build a Type 0 Routing Header with five Intermediate Nodes. */  
  
/* Determine the amount of memory required for the Routing  
 * Header and five intermediate nodes.  
 */  
rt_space = NU_Inet6_Rth_Space(IPV6_RTHDR_TYPE_0, 5);  
  
/* Allocate memory for the Routing Header. */  
NU_Allocate_Memory(&System_Memory, (VOID**) &ip6_rthdr, rt_space,  
                  NU_NO_SUSPEND);  
  
/* Initialize the ip6_rthdr structure. */  
ip6_rthdr = NU_Inet6_Rth_Init(ip6_rthdr, rt_space,  
                              IPV6_RTHDR_TYPE_0, 5);  
  
/* Add each intermediate node */  
NU_Inet6_Rth_Add(ip6_rthdr, I1);  
NU_Inet6_Rth_Add(ip6_rthdr, I2);  
NU_Inet6_Rth_Add(ip6_rthdr, I3);  
NU_Inet6_Rth_Add(ip6_rthdr, I4);  
NU_Inet6_Rth_Add(ip6_rthdr, D);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_GetAddr (IPv6)

This function returns a pointer to the IPv6 address specified by index (which must contain a value between 0 and one less than the value returned by NU\_Inet6\_Rth\_Segments()) in the Routing Header described by bp. An application should first call NU\_Inet6\_Rth\_Segments() to obtain the number of segments in the Routing Header.

### Usage

```
UINT8 *NU_Inet6_Rth_GetAddr (VOID *bp,  
                             INT  index);
```

### Arguments

- bp  
A pointer to the Routing Header.
- index  
The index value into the Routing Header of the IPv6 address to return.

### Return Values

- A pointer to the IPv6 address located in the Routing Header at the index value.  
Upon successful completion.
- NU\_NULL  
Otherwise

### Example

```
msghdr  msg;  
cmsghdr  *current_cmsg;  
INT      segments, i;  
UINT8    *rt_hdr_ptr;  
INT      *rt_hdr;  
  
/* Receive data via NU_Recvmsg(). */  
. . .  
  
/* Search for a route header in the received message. */  
for (current_cmsg = CMSG_FIRSTHDR(msg);  
     current_cmsg != NU_NULL;  
     current_cmsg = CMSG_NXTHDR(msg, current_cmsg))  
{  
    /* If this is a route header. */  
    if (current_cmsg->cmsg_type == IPV6_RTHDR)  
    {  
        /* Get a pointer to the route header. */  
        rt_hdr = (INT*)CMSG_DATA(current_cmsg);  
  
        /* Determine the number of segments in the route header. */  
        segments = NU_Inet6_Rth_Segments(rt_hdr);  
        /* Traverse the list of routes, getting a pointer to each one.  
        */  
        for (i = 0; i < segments; i++)
```

```

    {
        rt_hdr_ptr = NU_Inet6_Rth_GetAddr(rt_hdr, i);

        . . .
    }
}

```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_Init (IPv6)

This function initializes the buffer pointed to by bp to contain a Routing Header of the specified type and sets ip6r\_len based on the segments parameter. When the application uses ancillary data, the application must initialize any cmsghdr fields.

### Usage

```
VOID *NU_Inet6_Rth_Init (VOID    *bp,  
                        UINT32  bp_len,  
                        INT      type,  
                        INT      segments);
```

### Arguments

- bp  
A pointer to the buffer to initialize that will contain the Routing Header.
- bp\_len  
The total length of the final Routing Header as determined by NU\_Inet6\_Rth\_Space().
- type  
The type of Routing Header. Currently, only the Type 0 Routing Header is supported - IPV6\_RTHDR\_TYPE\_0.
- segments  
The number of segments that will be put in the Routing Header. For an IPv6 Type 0 Routing Header, the number of segments must be between 0 and 127, inclusive.

### Return Values

- The pointer to the buffer bp.  
Upon successful completion.
- NU\_NULL  
Otherwise

### Example

```
UINT32  rt_space;  
  
/** Build a Type 0 Routing Header with five Intermediate Nodes. */  
  
/* Determine the amount of memory required for the Routing  
 * Header and five intermediate nodes.  
 */  
rt_space = NU_Inet6_Rth_Space(IPV6_RTHDR_TYPE_0, 5);  
  
/* Allocate memory for the Routing Header. */  
NU_Allocate_Memory(&System_Memory, (VOID**) &ip6_rthdr, rt_space,  
                  NU_NO_SUSPEND);  
  
/* Initialize the ip6_rthdr structure. */
```



```
ip6_rthdr = NU_Inet6_Rth_Init(ip6_rthdr, rt_space,
                              IPV6_RTHDR_TYPE_0, 5);

/* Add each intermediate node. */
NU_Inet6_Rth_Add(ip6_rthdr, I1);
NU_Inet6_Rth_Add(ip6_rthdr, I2);
NU_Inet6_Rth_Add(ip6_rthdr, I3);
NU_Inet6_Rth_Add(ip6_rthdr, I4);
NU_Inet6_Rth_Add(ip6_rthdr, D);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_Reverse (IPv6)

This function takes a Routing Header extension header (pointed to by the first argument) and writes a new Routing Header that sends datagrams along the reverse of that route. The function reverses the order of the addresses and sets the segleft member in the new Routing Header to the number of segments. Both arguments are allowed to point to the same buffer (that is, the reversal can occur in place).

### Usage

```
INT NU_Inet6_Rth_Reverse (VOID *in,  
                          VOID *out);
```

### Arguments

- **in**  
A pointer to the source routing header.
- **out**  
A pointer to the target routing header.

### Return Values

- 0  
Upon successful completion.
- -1  
Otherwise.

### Description

Note that RFC 2460 states, “When an upper-layer protocol sends one or more packets in response to a received packet that included a Routing Header, the response packet(s) must not include a Routing Header that was automatically derived by “reversing” the received Routing Header UNLESS the integrity and authenticity of the received Source Address and Routing Header have been verified (for example, via the use of an authentication header in the received packet).”

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_Segments (IPv6)

This function returns the number of segments (addresses) contained in the Routing Header described by bp.

### Usage

```
INT NU_Inet6_Rth_Segments (const VOID *bp);
```

### Arguments

- bp  
A pointer to the Routing Header.

### Return Values

- The number of segments in the Routing Header.  
Upon successful completion.
- -1  
Otherwise

### Example

```
msghdr    msg;
cmsghdr    *current_cmsg;
INT         segments, i;
UINT8      *rt_hdr_ptr;
INT         *rt_hdr;

/* Receive data via NU_Recvmsg(). */
. . .

/* Search for a route header in the received message. */
for (current_cmsg = CMSG_FIRSTHDR(msg);
     current_cmsg != NU_NULL;
     current_cmsg = CMSG_NXTHDR(msg, current_cmsg))
{
    /* If this is a route header */
    if (current_cmsg->cmsg_type == IPV6_RTHDR)
    {
        /* Get a pointer to the route header. */
        rt_hdr = (INT*)CMSG_DATA(current_cmsg);

        /* Determine the number of segments in the route header. */
        segments = NU_Inet6_Rth_Segments(rt_hdr);

        /* Traverse the list of routes, getting a pointer to
         * each one.
         */
        for (i = 0; i < segments; i++)
        {
            rt_hdr_ptr = NU_Inet6_Rth_GetAddr(rt_hdr, i);

            . . .
        }
    }
}
```

```
    }  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#)

## NU\_Inet6\_Rth\_Space (IPv6)

This function returns the number of bytes required to hold a Routing Header of the specified type containing the specified number of segments (addresses). When the application uses ancillary data, it must pass the returned length to NU\_CMSG\_SPACE() to determine how much memory is needed for the ancillary data object (including the cmsghdr structure).

### Usage

```
INT NU_Inet6_Rth_Space (INT type,  
                        INT segments);
```

### Arguments

- **type**  
The type of Routing Header. Currently, only the Type 0 Routing Header is supported - IPV6\_RTHDR\_TYPE\_0.
- **segments**  
The number of segments that will be put in the Routing Header. For an IPv6 Type 0 Routing Header, the number of segments must be between 0 and 127, inclusive.

### Return Values

- The space required for the Routing Header.  
Upon successful completion.
- 0  
Otherwise, because either the type of the Routing Header is not supported by this implementation, or the number of segments is invalid for this type of Routing Header.

### Example

```
UINT32    rt_space;  
  
/** Build a Type 0 Routing Header with five Intermediate Nodes. */  
  
/* Determine the amount of memory required for the Routing  
 * Header and five intermediate nodes.  
 */  
rt_space = NU_Inet6_Rth_Space(IPV6_RTHDR_TYPE_0, 5);  
  
/* Allocate memory for the Routing Header. */  
NU_Allocate_Memory(&System_Memory, (VOID**)&ip6_rthdr, rt_space,  
                  NU_NO_SUSPEND);  
  
/* Initialize the ip6_rthdr structure. */  
ip6_rthdr = NU_Inet6_Rth_Init(ip6_rthdr, rt_space,  
                              IPV6_RTHDR_TYPE_0, 5);  
  
/* Add each intermediate node */  
NU_Inet6_Rth_Add(ip6_rthdr, I1);  
NU_Inet6_Rth_Add(ip6_rthdr, I2);  
NU_Inet6_Rth_Add(ip6_rthdr, I3);
```

```
NU_Inet6_Rth_Add(ip6_rthdr, I4);  
NU_Inet6_Rth_Add(ip6_rthdr, D);
```

## Related Topics

[Net API Functions](#)

[NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_GetAddr \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Reverse \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Find \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Get\\_Val \(IPv6\)](#)

[NU\\_Inet6\\_Opt\\_Next \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#)

[NU\\_Inet6\\_Rth\\_Segments \(IPv6\)](#)

## NU\_Inet\_NTOP (IPv4/IPv6)

This function converts an IP address into a readable ASCII string.

### Usage

```
STATUS NU_Inet_NTOP (INT  family,
                    VOID *src,
                    CHAR  *dst,
                    INT   size);
```

### Arguments

- family  
The family type of the IP address to convert. Either NU\_FAMILY\_IP for an IPv4 address or NU\_FAMILY\_IP6 for an IPv6 address.
- src  
Pointer to the IP address to convert.
- dst  
Pointer to the memory into which to put the converted ASCII string representation of the IP address.
- size  
The size of the memory provided for the converted ASCII string.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
Either the pointers passed in were invalid or the size of memory provided was insufficient for the conversion.

### Example

```
UINT8 ip_dest[4] = {192, 200, 100, 1};
CHAR  char_buff[16];
.
.
.

/* Convert the destination IP addr to ASCII. */
status = NU_Inet_NTOP(NU_FAMILY_IP, ip_dest, char_buff, 16);
```

### Related Topics

[Net API Functions](#)

[NU\\_Inet\\_PTON \(IPv4/IPv6\)](#)



## NU\_Inet\_PTON (IPv4/IPv6)

This function converts an ASCII string into a network byte-order IP address.

### Usage

```
STATUS NU_Inet_PTON (INT  family,
                    CHAR  *src,
                    VOID  *dst);
```

### Arguments

- **family**  
The family type of the IP address to convert. Either NU\_FAMILY\_IP for an IPv4 address of NU\_FAMILY\_IP6 for an IPv6 address.
- **src**  
Pointer to the ASCII string that will be converted into network byte-order IP address.
- **dst**  
Pointer to the memory where the IP address will be written.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - **NU\_INVALID\_PARM**  
Either the pointers passed in were invalid or the size of memory provided was insufficient for the conversion.

### Example

```
CHAR    char_buff[40];
UINT8   ip_addr[16];

/* Store an ASCII IPv6 address in char_buff. */
.
.
.
/* Convert the received ASCII string into an IP address. */
STATUS NU_Inet_PTON(NU_FAMILY_IP6, char_buff, ip_addr);
```

### Related Topics

[Net API Functions](#)

[NU\\_Inet\\_NTOP \(IPv4/IPv6\)](#)

## NU\_Init\_Devices (IPv4/IPv6)

This function is responsible for initializing all networking interfaces that will be used by Nucleus NET. You set up the interfaces at the Application Layer based on the node's requirements. This routine is called at most once per network-enabled interface in the system.

### Usage

```
STATUS NU_Init_Devices (NU_DEVICE *devices,  
                        INT16      dev_count);
```

### Arguments

- **devices**  
Array of device information structures. The structure **NU\_DEVICE** is defined in the [Net Data Structures](#) section.
- **dev\_count**  
Number of device structures in the device array.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code, an interface-specific error code, or the following Nucleus NET error code:
- **NU\_INVALID\_PARM**  
Either the pointer to devices is **NU\_NULL** or dev\_count is less than or equal to zero.

### Example - IPv4

```
NU_DEVICE devices[1];  
  
devices[0].dv_name = NU_Argv[1];  
devices[0].dv_hw.ether.dv_irq = 0; /* Unused for this interface. */  
devices[0].dv_hw.ether.dv_io_addr = 0; /* Unused for this interface. */  
devices[0].dv_hw.ether.dv_shared_addr = 0;  
devices[0].dv_init = VDRV_Init;  
  
/* Set up the IPv4 address and subnet mask. */  
memcpy (devices[0].dv_ip_addr, ip_addr, 4);  
memcpy (devices[0].dv_subnet_mask, subnet, 4);  
  
devices[0].dv_flags = 0;  
  
. . .  
  
status = NU_Init_Devices(devices, 1);  
  
/* If status is NU_SUCCESS, then the devices specified were  
   successfully initialized. */
```

### Example - IPv6

```
NU_DEVICE devices[1];
```

```

devices[0].dv_name = NU_Argv[1];
devices[0].dv_hw.ether.dv_irq = 0; /* Unused for this interface. */
devices[0].dv_hw.ether.dv_io_addr = 0; /* Unused for this interface. */
devices[0].dv_hw.ether.dv_shared_addr = 0;
devices[0].dv_init = VDRV_Init;

/* Set up the IPv4 address and subnet mask. */
memcpy (devices[0].dv_ip_addr, ip_addr, 4);
memcpy (devices[0].dv_subnet_mask, subnet, 4);

/* Set the flag to indicate this is an IPv6 interface and should
 * be used as the primary interface.
 */
devices[0].dv_flags =
    (DV6_IPV6 | DV6_PRIMARY_INT);
. . .

status = NU_Init_Devices(devices, 1);

/* If status is NU_SUCCESS, then the devices specified were
   successfully initialized. Note that an IPv6 address is not
   assigned to the device by the application at initialization. A
   link-local address is configured dynamically at initialization
   along with a globally unique IPv6 address(es) if there are any
   IPv6 routers on the network configured to assign IPv6 prefixes. */

```

## Related Topics

[Net API Functions](#)

[NU\\_Init\\_Net \(IPv4/IPv6\)](#)

## NU\_Init\_Net (IPv4/IPv6)

This is the first function that is called before utilizing Nucleus NET. It is responsible for setting up the internal data structures for the network stack and allocating any resources required by Nucleus NET. This function must only be called once.

### Usage

```
STATUS NU_Init_Net (NU_NET_INIT_DATA *init_struct);
```

### Arguments

- `init_struct`

A pointer to the data structure containing the initialization data. The `net_buffered_mem` member holds a pointer to the memory pool to use to allocate the NET buffers. The `net_internal_mem` member holds a pointer to the memory pool to use for all other Nucleus NET memory allocations.

The `net_buffered_mem` member of the `init_struct` can be used to specify a memory pool of non-cached memory. This is useful because Nucleus NET drivers share the pool of memory buffers with the stack. On some architectures, the device will be given a pointer to a memory buffer and will write received data and status information directly to the buffer. On such architectures, it is necessary to frequently flush the cache before accessing buffers that have just been returned by the driver to the stack. As an alternative, a non-cached memory pool can be specified from which to allocate Nucleus NET buffers and data structures that can be accessed by both hardware and the driver. If such a non-cached memory pool is not needed or desired, then any memory pool will do, including the `System_Memory` pool.

### Return Values

- `NU_SUCCESS`

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- `NU_INVALID_PARM`

One of the input parameters is `NU_NULL`.

- `NU_MEM_ALLOC`

One of the memory pools have been exhausted before initialization could complete.

- `NU_INVALID`

A general-purpose error condition. This generally indicates that a required resource (task, semaphore, and so on.) could not be created.

- `NU_DHCP_INIT_FAILED`

Failed to initialize the DHCP module.

## Example

```
STATUS status; /* The status that will be returned. */
NU_Memory_Pool System_Memory;
NU_Memory_Pool DEMOI_NET_Memory_Pool;
NU_NET_INIT_DATA init_struct;

/* Create a system memory pool that will be used to allocate
   task stacks, queue areas, etc. */
status = NU_Create_Memory_Pool(&System_Memory, "SYSMEM",
                               first_available_memory, 400000,
                               50, NU_Fifo);

if (status != NU_SUCCESS)
{
    printf("Could not create the System Memory Pool.\n");
}

/* Create the memory pool for use by NET. It will contain
   the NET memory buffers. */
status = NU_Create_Memory_Pool(&DEMOI_NET_Memory_Pool, "NETMEM",
                               (unsigned char *)next_available_memory, 30000, 50,
                               NU_Fifo);

if (status != NU_SUCCESS)
{
    printf("Could not create the NET Buffer Memory Pool.\n");
}

init_struct.net_buffered_mem = &DEMOI_NET_Memory_Pool;
init_struct.net_internal_mem = &System_Memory;

status = NU_Init_Net(&init_struct);

/* Was the stack successfully initialized? */
if (status != NU_SUCCESS)
{
    printf("Could not initialize Nucleus NET.\n");
}
```

## Related Topics

[NET User Management](#)

[NU\\_Init\\_Devices \(IPv4/IPv6\)](#)

## NU\_Ioctl (IPv4/IPv6)

This function is responsible for performing specialized functions on an interface or other object.

### Usage

```
STATUS NU_Ioctl (INT                optname,
                SCK_IOCTL_OPTION *option,
                INT                optlen);
```

### Arguments

- **optname**

Specifies an option. Valid values for this parameter are:

SIOCGIFADDR	SIOCGIFNETMASK	SIOCSETVLANPRIO
SIOCGIFHWADDR	SIOCGIFDSTADDR	SIOCSPHYSADDR
SIOCIFREQ	SIOCGARP	SIOCSARP
SIOCGIFADDR_IN6	SIOCGETVLAN	SIOCDDARP
SIOCSETVLAN	SIOCSIFADDR	SIOCGIFDSTADDR_IN6
SIOCGHWCAP	SIOCSIFHWADDR	FIONREAD
SIOCGETVLANPRIO	SIOCSHWOPTS	SIOCICMPLIMIT

If an unrecognized value is specified, the IOCTL command will be issued directly to the network interface specified by option->s\_ret.s\_ipaddr.

[Table 2-37](#) lists the supported options for the NU\_Ioctl optname argument.

**Table 2-37. NU\_Ioctl Options**

Option	Family	Description
SIOCGIFADDR	IPv4	Get the interface address associated with a device.
SIOCGIFHWADDR	IPv4/IPv6	Get the MAC address of a network interface by using its interface name. This IOCTL can be issued in both the link-down or link-up state.
SIOCSIFADDR	IPv4	Set the interface address for a device.
SIOCSIFHWADDR	IPv4/IPv6	Set the MAC address of a network interface by using its interface name. This IOCTL is to be used when the network interface is in a link-down state.
SIOCGIFDSTADDR	IPv4	Get the point-to-point address associated with a device.
SIOCSPHYSADDR	IPv4/IPv6	Set the physical address associated with a device.

Table 2-37. NU\_ioctl Options (cont.)

Option	Family	Description
SIOCSARP	IPv4	Create or modify an ARP Cache entry.
SIOCDELARP	IPv4	Delete an ARP Cache entry.
SIOCGARP	IPv4	Get an ARP Cache entry.
SIOCICMPLIMIT	IPv4	Set the rate-limiting interval for outgoing ICMP error messages.
SIOCIFREQ	IPv4/IPv6	This option invokes a call directly to the ioctl function of the device specified in the s_optval parameter of the SCK_IOCTL_OPTION. The device-specific option is stored in the dvr_optname parameter of the DV_REQ parameter of the SCK_IOCTL_OPTION.
SIOCGIFADDR_IN6	IPv6	Get the IPv6 interface address of best scope associated with a device. If an address of global scope is present, it is returned; otherwise, the link-local address is returned.
SIOCGIFDSTADDR_IN6	IPv6	Get the point-to-point IPv6 address associated with the device on the other side of the PPP link.
FIONREAD	IPv4/IPv6	Return the number of bytes of data pending to be received on the socket. s_optval is set by the application to point to the socket descriptor. s_ret.sck_bytes_pending is filled in by the routine with the number of bytes of data pending on the socket.
SIOCSETVLAN	IPv4	Set the VLAN Group ID association.
SIOCGETVLAN	IPv4	Get the VLAN Group ID association.
SIOCGHWCAP	IPv4/IPv6	Get the hardware offloading capabilities of the interface.
SIOCSHWOPTS	IPv4/IPv6	Set the hardware offloading options of the interface.
SIOCGETVLANPRIO	IPv4/IPv6	Get the VLAN priority associated with an interface.

**Table 2-37. NU\_ioctl Options (cont.)**

Option	Family	Description
SIOCSETVLANPRIO	IPv4/IPv6	Set the VLAN priority associated with an interface.
SIOCGIFNETMASK	IPv4	Get the subnet mask associated with an IP address for an interface. If no IP address is specified, get the subnet mask associated with the first IP address entry on the interface.

- **option**  
 Return pointer for option status. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.
- **optlen**  
 Specifies the size in bytes of the location pointed to by option.

### Return Values

- **NU\_SUCCESS**  
 Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- **NU\_INVALID**  
 The value specified in the optlen parameter was not large enough to store the option status.
  - **NU\_INVALID\_PARM**  
 The device specified by the option parameter is invalid.
  - **NU\_INVALID\_OPTION**  
 The value specified in the optname parameter is unrecognized, and there is no interface in the system matching the IP address specified in option->s\_ret.s\_ipaddr to which to issue the IOCTL command directly.
  - **NU\_INVALID\_SOCKET**  
 The socket index passed into the function is invalid.

### Example

```
DEV_DEVICE devices[1];
NU_IOCTL_OPTION option;
IP_MREQ mgroup;

. . .

/* This example shows how to find the IP address attached to a
```



```

    particular device. */

/* Point to the name of the device of which you want to know the IP
   address. */
option.s_optval = devices[0].dv_name;

/* Call NU_Ioctl to get the IP address. */
if (NU_Ioctl(SIOCGIFADDR, &option, sizeof(option)) != NU_SUCCESS)
{
    printf("Could not get the IP address.\n");
}

/* Copy the retrieved IP address. */
memcpy (&mgroup.sck_inaddr, option.s_ret.s_ipaddr, 4);

```

### Example - SIOCIFREQ

```

DEV_DEVICE devices[1];
NU_IOCTL_OPTION option;

. . .

/* This example shows how to issue an ioctl command not supported by the
   NET stack directly to the device. */

/* Set up the name of the device. */
option.s_optval = devices[0].dv_name;

/* Set up the option that will be recognized by the device's
   * ioctl routine.
   */
option.s_ret.s_dvreq.dvr_optname = DEVICE_OPTION;

/* Call NU_Ioctl to process the request. */
if (NU_Ioctl(SIOCIFREQ, &option, sizeof(option)) != NU_SUCCESS)
{
    printf("Interface failed to process IOCTL request.\n");
}

```

### Example - SIOCSIFHWADDR

As for any Ioctl, This Ioctl should also be used along with an instance of NU\_IOCTL\_OPTION. The name of the network interface who's MAC we want to update should be assigned to NU\_IOCTL\_OPTION. s\_optval and the new MAC address should be assigned to NU\_IOCTL\_OPTION. s\_ret.mac\_address[].

```

SCK_IOCTL_OPTION option;
UINT8 mac_address[6] = {0x00, 0x10, 0x78, 0xA3, 0x4C, 0x7F};

.
.
.

/* Populate the IOCTL structure. */
option.s_optval = "eth0";
memcpy(option.s_ret.mac_address, mac_address, 6);

```

```
/* Bring down the interface. */
if (NU_Ethernet_Link_Down("eth0") == NU_SUCCESS)
{
    /* Change the MAC address for this interface. */
    if (NU_Ioctl_SIOCSIFHWADDR(&option) == NU_SUCCESS)
    {
        printf("MAC address is updated\r\n");
    }
    else
    {
        printf("Unable to change MAC address\r\n");
    }
}

/* Bring up the interface. */
NU_Ethernet_Link_Up("eth0");
```

### Example - SIOCGIFHWADDR

As for any Ioctl, This Ioctl should also be used along with an instance of NU\_IOCTL\_OPTION. The name of the network interface whose MAC we want to update should be assigned to NU\_IOCTL\_OPTION.s\_optval.

```
STATUS status;
CHAR if_name[] = "eth0";
NU_IOCTL_OPTION ioctl_opt;

/* Initialize the IOCTL option to zero. */
memset(&ioctl_opt, 0, sizeof(NU_IOCTL_OPTION));

/* Name of the network interface whose MAC address we need. */
ioctl_opt.s_optval = (UINT8*)if_name;

/* Call NU_Ioctl to get the current MAC Address. */
status = NU_Ioctl(SIOCGIFHWADDR, &ioctl_opt, sizeof(ioctl_opt));

/* Check if we got the MAC */
if (status == NU_SUCCESS)
{
    /* Print the MAC Address */
    printf("%.2X-%.2X-%.2X-%.2X-%.2X-%.2X",
        ioctl_opt.s_ret.mac_address[0],
        ioctl_opt.s_ret.mac_address[1],
        ioctl_opt.s_ret.mac_address[2],
        ioctl_opt.s_ret.mac_address[3],
        ioctl_opt.s_ret.mac_address[4],
        ioctl_opt.s_ret.mac_address[5]);
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGETVLAN \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSETVLAN \(IPv4\)](#)

[NU\\_Ioctl\\_FIONREAD \(IPV4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCDARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFNETMASK \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGETVLANPRIO \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCIFREQ \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSETVLANPRIO \(IPv4/IPv6\)](#)

## NU\_Ioctl\_FIONREAD (IPV4/IPv6)

This function is responsible for returning the number of bytes of data pending on the socket to be read.

### Usage

```
STATUS NU_Ioctl_FIONREAD (SCK_IOCTL_OPTION *option);
```

### Arguments

- option  
The parameter s\_optval contains the socket index for which to retrieve the number of bytes pending. The parameter s\_ret.sck\_bytes\_pending will be filled in with the number of bytes pending on the socket. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.
  - NU\_INVALID\_SOCKET  
The socket is not valid in the system.

### Example

```
SCK_IOCTL_OPTION option;  
INT socketd;  
  
/* Build a socket. */  
. . .  
  
/* Set the socket ID.*/  
option.s_optval = (UINT8*)&socketd;  
  
/* Call NU_Ioctl_FIONREAD to get the number of bytes pending on the  
 * socket.  
 */  
if (NU_Ioctl_FIONREAD(&option) != NU_SUCCESS)  
{  
    printf("Could not get number of bytes on the socket.\n");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCDARP (IPv4)

This function is responsible for deleting an entry from the IPv4 ARP cache.

### Usage

```
STATUS NU_Ioctl_SIOCDARP (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_ret.arp_request.arp_pa.sck_data` contains the IPv4 address of the ARP cache entry to delete. The [SCK\\_IOCTL\\_OPTION](#) is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

One of the input parameters is invalid.

### Example

```
SCK_IOCTL_OPTION option;
UINT8 ip_addr[IP_ADDR_LEN] = {192,168,50,61};

. . .

/* Copy the IP address of the ARP Cache entry to delete.
 */
memcpy(option->s_ret.arp_request.arp_pa.sck_data, ip_addr,
        IP_ADDR_LEN);

/* Call NU_Ioctl_SIOCDARP to delete the arp cache entry. */
if (NU_Ioctl_SIOCDARP(&option) != NU_SUCCESS)
{
    printf("Could not delete the ARP Cache entry.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCGARP (IPv4)

This function is responsible for retrieving an entry from the IPv4 ARP cache.

### Usage

```
STATUS NU_Ioctl_SIOCGARP (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_ret.arp_request.arp_pa.sck_data` contains the IPv4 address of the ARP cache entry to retrieve. The parameter `s_ret.arp_request.arp_ha.sck_data` is filled in with the MAC address of the entry. The parameter `s_ret.arp_request.arp_flags` is filled in with the ARP cache entry flags. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

One of the input parameters is invalid.

### Example

```
SCK_IOCTL_OPTION option;
UINT8 ip_addr[IP_ADDR_LEN] = {192,168,50,61};

. . .

/* Copy the IP address of the ARP Cache entry whose MAC
 * address you want to retrieve.
 */
memcpy(option->s_ret.arp_request.arp_pa.sck_data, ip_addr,
        IP_ADDR_LEN);

/* Call NU_Ioctl_SIOCGARP to get the arp cache entry. */
if (NU_Ioctl_SIOCGARP(&option) != NU_SUCCESS)
{
    printf("Could not get the ARP Cache entry.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCDBARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCGIFADDR (IPv4)

This function is responsible getting the IPv4 address associated with an interface.

### Usage

```
STATUS NU_Ioctl_SIOCGIFADDR(SCK_IOCTL_OPTION *option,  
                             INT optlen);
```

### Arguments

- option  
The parameter s\_optval contains the name of the device for which the IPv4 address is being requested. The parameter s\_ret.s\_ipaddr will be filled in with the requested IPv4 address. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.
- optlen  
Specifies the size in bytes of the location pointed to by option.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Example

```
DEV_DEVICE      devices[1];  
SCK_IOCTL_OPTION option;  
  
. . .  
  
/* Point to the name of the device of which you want to know  
 * the IP address.  
 */  
option.s_optval = devices[0].dv_name;  
  
/* Call NU_Ioctl_SIOCGIFADDR to get the IP address. */  
if (NU_Ioctl_SIOCGIFADDR(&option, sizeof(option)) != NU_SUCCESS)  
{  
    printf("Could not get the IP address.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)



## NU\_Ioctl\_SIOCGIFADDR\_IN6 (IPv6)

This function is responsible for getting the IPv6 address associated with an interface. If a globally unique IPv6 address does not exist on the device, the link-local address is returned. If no link-local address exists, an error is returned.

### Usage

```
STATUS NU_Ioctl_SIOCGIFADDR_IN6 (SCK_IOCTL_OPTION *option,
                                INT optlen);
```

### Arguments

- option  
 The parameter s\_optval contains the name of the interface of which to retrieve the IPv6 address. The parameter s\_ret.s\_ipaddr will be filled in with the IPv6 address requested. The parameters s\_optval\_octet will contain the IPv6 prefix length. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.
- optlen  
 Specifies the size in bytes of the location pointed to by option.

### Return Values

- NU\_SUCCESS  
 Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
 The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Point to the name of the device of which you want to know
 * the IPV6 address.
 */
option.s_optval = devices[0].dv_name;

/* Call NU_Ioctl_SIOCGIFADDR_IN6 to get the IPV6
 * address.
 */
if (NU_Ioctl_SIOCGIFADDR_IN6(&option, sizeof(option)) != NU_SUCCESS)
{
    printf("Could not get the IPv6 address.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

## NU\_Ioctl\_SIOCGIFDSTADDR (IPv4)

This function is responsible for getting the IPv4 address of the foreign side of a PPP link.

### Usage

```
STATUS NU_Ioctl_SIOCGIFDSTADDR (SCK_IOCTL_OPTION *option,  
                                INT optlen);
```

### Arguments

- option  
The parameter `s_optval` contains the name of the device for which the foreign IPv4 address is being requested. The parameter `s_ret.s_ipaddr` will be filled in with the requested foreign IPv4 address. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.
- optlen  
Specifies the size in bytes of the location pointed to by option.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Example

```
DEV_DEVICE      devices[1]; /* A PPP device */  
SCK_IOCTL_OPTION option;  
  
. . .  
  
/* Point to the name of the PPP device. */  
option.s_optval = devices[0].dv_name;  
  
/* Call NU_Ioctl_SIOCGIFDSTADDR to get the IP address of the other  
 * end of the PPP link.  
 */  
if (NU_Ioctl_SIOCGIFDSTADDR(&option, sizeof(option)) != NU_SUCCESS)  
{  
    printf("Could not get IP address of PPP link.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

## NU\_Ioctl\_SIOCGIFDSTADDR\_IN6 (IPv6)

This function is responsible for getting the IPv6 address of the foreign side of a PPP link.

### Usage

```
STATUS NU_Ioctl_SIOCGIFADDR_IN6 (SCK_IOCTL_OPTION *option,  
                                INT optlen);
```

### Arguments

- option  
The parameter `s_optval` contains the name of the interface of which to retrieve the foreign IPv6 address. The parameter `s_ret.s_ipaddr` will be filled in with the foreign IPv6 address. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.
- optlen  
Specifies the size in bytes of the location pointed to by option.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:
  - NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Example

```
DEV_DEVICE      devices[1];  
SCK_IOCTL_OPTION option;  
  
. . .  
  
/* Point to the name of the device. */  
option.s_optval = devices[0].dv_name;  
  
/* Call NU_Ioctl_SIOCGIFDSTADDR to get the IP address of the other  
   end of a PPP link. */  
if (NU_Ioctl_SIOCGIFADDR_IN6(&option, sizeof(option)) !=  
    NU_SUCCESS)  
{  
    printf("Could not get IPv6 address of PPP link.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCSPHYSADDR \(IPv4\)](#)

## NU\_Ioctl\_SIOCGIFHWADDR

This API gets the MAC address of the network interface by its name. This API can be used in both the link-down or link-up state.

### Usage

```
STATUS NU_Ioctl_SIOCGIFHWADDR(SCK_IOCTL_OPTION *option);
```

### Arguments

- option  
Should contain the name of an interface whose MAC has to be fetched, the name must be assigned to option->s\_optval.  
Will also contain the returned MAC address, and it should be assigned to:  
option->s\_ret.mac\_address[].

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
No device exists with the specified interface name.

### Example

```
STATUS status;
CHAR if_name[] = "eth0";
NU_IOCTL_OPTION ioctl_opt;

/* Initialize the IOCTL option to zero. */
memset(&ioctl_opt, 0, sizeof(NU_IOCTL_OPTION));

/* Name of the network interface whose MAC address we need. */
ioctl_opt.s_optval = (UINT8*)if_name;

/* Call NU_Ioctl to get the current MAC Address. */
status = NU_Ioctl_SIOCGIFHWADDR (&ioctl_opt);

/* Check if we got the MAC */
if (status == NU_SUCCESS)
{
    /* Print the MAC Address */
    printf("%.2X-%.2X-%.2X-%.2X-%.2X-%.2X",
           ioctl_opt.s_ret.mac_address[0],
           ioctl_opt.s_ret.mac_address[1],
           ioctl_opt.s_ret.mac_address[2],
           ioctl_opt.s_ret.mac_address[3],
           ioctl_opt.s_ret.mac_address[4],
           ioctl_opt.s_ret.mac_address[5]);
}
```

## Related Topics

[Net API Functions](#)



## NU\_Ioctl\_SIOCGIFNETMASK (IPv4)

This function gets the subnet mask associated with an IP address for a specified network interface. If no IP address is specified, this function gets the subnet mask associated with the first IP address in the list of addresses on the interface.

### Usage

```
STATUS NU_Ioctl_SIOCGIFNETMASK (SCK_IOCTL_OPTION *option,
                                INT                opt_len);
```

### Arguments

- **option**  
 A pointer to the data structure that holds the name of the interface and the IP address for which to retrieve the subnet mask.  
 The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section. Members related to this function call are: `s_optval` and `vlan_prio`.
- **opt\_len**  
 Specifies the size in bytes of the location pointed to by `option`.

### Return Values

- **NU\_SUCCESS**  
 Upon successful completion, the `s_ret.s_ipaddr` member of the [SCK\\_IOCTL\\_OPTION](#) data structure holds the subnet mask associated with the IP address. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code.
- **NU\_INVALID\_PARM**  
 One of the input parameters is invalid, the interface does not exist in the system, or the specified IP address does not exist on the interface.

### Example

```
SCK_IOCTL_OPTION  option;
STATUS            status;
UINT8             ip_addr[] = {192, 168, 1, 98};

/* Set the pointer to the interface name. */
option.s_optval = "eth0";

/* Set the IP address for which to retrieve the subnet mask. */
memcpy(option.s_ret.s_ipaddr, ip_addr, IP_ADDR_LEN);

/* Make the function call to retrieve the subnet mask associated with
 * the IP address on the specified interface.
 */
status = NU_Ioctl_SIOCGIFNETMASK(&option, sizeof(SCK_IOCTL_OPTION));

if (status != NU_SUCCESS)
{
```

```
        printf("Error at call to NU_Ioctl_SIOCGIFNETMASK.\n");  
    }
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCGETVLAN (IPv4)

This function is responsible getting the VLAN ID associated with the interface.

### Usage

```
STATUS NU_Ioctl_SIOCGETVLAN (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_optval` contains the name of the device of which to get the VLAN ID. The parameter `s_ret.vlan_id` contains the retrieved VLAN ID. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE      devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Set the device name. */
option->s_optval = devices[0].dv_name;

/* Call NU_Ioctl_SIOCGETVLAN to retrieve the VLAN ID. */
if (NU_Ioctl_SIOCGETVLAN(&option) != NU_SUCCESS)
{
    printf("Could not get the VLAN ID.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSETVLAN \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCGETVLANPRIO (IPv4/IPv6)

This function gets the VLAN priority associated with an interface.

### Usage

```
STATUS NU_Ioctl_SIOCGETVLANPRIO (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

A pointer to the data structure that holds the name of the interface for which to get the VLAN priority.

The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section. The members of the structure related to this function call are `s_optval` and `vlan_prio`.

### Return Values

- NU\_SUCCESS

Upon successful completion, the `s_ret.vlan_prio` member of the [SCK\\_IOCTL\\_OPTION](#) data structure will hold the VLAN priority associated with the interface. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error code.

- NU\_INVALID\_PARM

One of the input parameters is invalid.

### Example

```
SCK_IOCTL_OPTION    option;
STATUS              status;

/* Set the pointer to the interface name. */
option.s_optval = "eth0";

/* Make the function call to retrieve the VLAN priority value for
 * the specified interface.
 */
status = NU_Ioctl_SIOCGETVLANPRIO(&option);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Ioctl_SIOCGETVLANPRIO.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCSETVLANPRIO \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCGHWCAP (IPv4/IPv6)

This function is responsible for getting the hardware offloading capabilities of an interface.

### Usage

```
STATUS NU_Ioctl_SIOCGHWCAP (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_optval` contains the name of the device for which the hardware offloading options are being retrieved. The parameter `s_ret.s_dvreq.dvr_flags` contains the hardware offloading capabilities of the interface. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code of the following Nucleus NET error code:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE      devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Set a pointer to the name of the interface for which to retrieve the
 * set of supported hardware offloading options.
 */
option->s_optval = devices[1].dv_name;

/* Call NU_Ioctl_SIOCGHWCAP to determine which hardware offloading
 * options the interface supports.
 */
if (NU_Ioctl_SIOCGHWCAP(&option) != NU_SUCCESS)
{
    printf("Could not get hw offloading options.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCICMPLIMIT

This function configures the rate-limiting value of outgoing ICMP error messages at run-time for the specified interface. If less than `max_errors` ICMP error messages have been transmitted on the interface over the specified interval, a new ICMP error message may be transmitted out the interface; otherwise, no new ICMP error messages may be transmitted until the interval expires. The interval value is set to `ICMP_ERROR_MSG_RATE_LIMIT`, and `max_errors` set to `ICMP_ERROR_MSG_COUNT_RATE_LIMIT` by default when the interface is initialized. These macros can be configured at compile-time in the file *net\_cfg.h*.

### Usage

```
STATUS NU_Ioctl_SIOCICMPLIMIT (CHAR    *dev_name,  
                               UINT8    max_errors,  
                               UINT32   interval)
```

### Arguments

- `dev_name`  
Pointer to the name of the interface for which to configure the ICMP rate-limiting value.
- `max_errors`  
The number of ICMP error messages to permit over a certain interval.
- `interval`  
The interval in hardware ticks over which to limit the number of ICMP error messages transmitted over the specified interface.

### Return Values

- `NU_SUCCESS`  
Indicates successful completion of the service.
- `NU_INVALID_PARM`  
One of the input parameters is invalid.

### Example

```
STATUS      status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Configure the interface to allow two outgoing ICMP error messages  
 * per second.  
 */  
status = NU_Ioctl_SIOCICMPLIMIT("eth0", 2,  
                                1 * SCK_Ticks_Per_Second);  
  
if (status == NU_SUCCESS)
```

```
{  
    . . .  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCIFREQ (IPv4/IPv6)

This function is responsible for issuing an IOCTL command directly to the device.

### Usage

```
STATUS NU_Ioctl_SIOCIFREQ (SCK_IOCTL_OPTION *option);
```

### Arguments

- option  
The parameter `s_optval` contains the name of the interface to which to issue the command. The parameter `s_ret.s_dvreq.dvr_optname` contains the interface-specific command to issue. The parameter `s_ret.s_dvreq` contains any additional data associated with the command. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or an interface-specific error code.
- NU\_INVALID\_PARM  
One of the input parameters is invalid.

### Example

```
DEV_DEVICE      devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Set up the name of the device. */
option.s_optval = devices[0].dv_name;

/* Set up the option that will be recognized by the device's
 * ioctl routine.
 */
option.s_ret.s_dvreq.dvr_optname = DEVICE_OPTION;

/* Call NU_Ioctl to process the request. */
if (NU_Ioctl_SIOCIFREQ(&option) != NU_SUCCESS)
{
    printf("Interface failed to process IOCTL request.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)



## NU\_Ioctl\_SIOCSARP (IPv4)

This function is responsible for adding/modifying an entry to the IPv4 ARP cache.

### Usage

```
STATUS NU_Ioctl_SIOCSARP (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_ret.arp_request.arp_pa.sck_data` contains the IPv4 address of the ARP cache entry to update. The parameter `s_ret.arp_request.arp_ha.sck_data` contains the new MAC address of the entry. The parameter `s_ret.arp_request.arp_flags` contains new flags to set or 0 to leave the ARP cache entry flags unchanged. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

- NU\_NO\_MEMORY

A matching ARP entry could not be found, and a new entry could not be created.

### Example

```
SCK_IOCTL_OPTION option;
UINT8 ip_addr[IP_ADDR_LEN] = {192,168,50,61};
UINT8 hw_addr[DADDLEN] = {0x00, 0x0C, 0x00, 0x0A, 0x00, 0x0B};
. . .

/* Copy the IP address. */
memcpy(option.s_ret.arp_request.arp_pa.sck_data, ip_addr, IP_ADDR_LEN);

/* Copy the MAC address. */
memcpy(option.s_ret.arp_request.arp_ha.sck_data, hw_addr, DADDLEN);

/* Do not modify the flags of the entry. */
option.s_ret.arp_request.arp_flags = NU_NULL;

/* Call NU_Ioctl_SIOCSARP to add/modify the arp cache. */
if (NU_Ioctl_SIOCSARP(&option) != NU_SUCCESS)
{
    printf("Could not modify ARP Cache.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCDARP \(IPv4\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCASHWOPTS (IPv4/IPv6)

This function is responsible for setting the hardware offloading options of an interface. The interface must support the specified options in order for them to go into effect.

### Usage

```
STATUS NU_Ioctl_SIOCASHWOPTS (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section. The parameter `s_optval` contains the name of the device for which the hardware offloading options are being set. The parameter `s_ret.s_dvreq.dvr_scmd` specifies whether to enable or disable the provided options. `DEV_DISABLE_HW_OPTIONS` disables the specified options. `DEV_ENABLE_HW_OPTIONS` enables the specified options. The parameter `s_ret.s_dvreq.dvr_pcmd` contains the new hardware offloading options:

**HW\_TX\_IP4\_CHKSUM**

Enable hardware checksum offloading of the IP checksum in outgoing IPv4 packets.

**HW\_RX\_IP4\_CHKSUM**

Enable hardware checksum offloading of the IP checksum in incoming IPv4 packets.

**HW\_TX\_TCP\_CHKSUM**

Enable hardware checksum offloading of the TCP checksum in outgoing IPv4 packets.

**HW\_RX\_TCP\_CHKSUM**

Enable hardware checksum offloading of the TCP checksum in incoming IPv4 packets.

**HW\_TX\_TCP6\_CHKSUM**

Enable hardware checksum offloading of the TCP checksum in outgoing IPv6 packets.

**HW\_RX\_TCP6\_CHKSUM**

Enable hardware checksum offloading of the TCP checksum in incoming IPv6 packets.

**HW\_TX\_UDP\_CHKSUM**

Enable hardware checksum offloading of the UDP checksum in outgoing IPv4 packets.

**HW\_RX\_UDP\_CHKSUM**

Enable hardware checksum offloading of the UDP checksum in incoming IPv4 packets.

**HW\_TX\_UDP6\_CHKSUM**

Enable hardware checksum offloading of the TCP checksum in outgoing IPv6 packets.

#### HW\_RX\_UDP6\_CHKSUM

Enable hardware checksum offloading of the TCP checksum in incoming IPv6 packets.

#### HW\_TX\_VLAN

Enable hardware offloading for transmission of VLAN packets.

#### HW\_RX\_VLAN

Enable hardware offloading for reception of VLAN packets.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code, driver-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
NU_DEVICE      devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Set a pointer to the name of the interface for which to
 * set the hardware offloading options.
 */
option->s_optval = devices[1].dv_name;

/* Set the flag to enable the specified options. */
option.s_ret.s_dvreq.dvr_scmd = DEV_ENABLE_HW_OPTIONS;

/* Instruct the hardware to verify the IPv4 checksum on incoming
 * packets and compute the IPv4 checksum on outgoing packets.
 */
option.s_ret.s_dvreq.dvr_pcmd =
    HW_TX_IP4_CHKSUM | HW_RX_IP4_CHKSUM;

/* Call NU_Ioctl_SIOCGHWCAP to set the offloading options. */
if (NU_Ioctl_SIOCGHWCAP(&option) != NU_SUCCESS)
{
    printf("Could not set hw offload options.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCSIFADDR (IPv4)

This function is responsible for setting the IPv4 address associated with an interface.

### Usage

```
STATUS NU_Ioctl_SIOCSIFADDR (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_ret.s_ipaddr` contains the IPv4 address of the device for which the IPv4 address is being set. The parameter `s_optval` contains the new IPv4 address. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE      devices[1];
SCK_IOCTL_OPTION option;
UINT8   ip_addr[IP_ADDR_LEN] = {192,168,60,50};

. . .

/* Copy the old IP address of the device of which to set
 * the IP address.
 */
memcpy(option->s_ret.s_ipaddr, devices[0].dv_ip_addr,
        IP_ADDR_LEN);

/* Point to the new IP address. */
option.s_optval = ip_addr;

/* Call NU_Ioctl_SIOCSIFADDR to set the IP address. */
if (NU_Ioctl_SIOCSIFADDR (&option) != NU_SUCCESS)
{
    printf("Could not set IP address.\n");
}
```

Related Topics

- Net API Functions
- NU\_Ioctl\_SIOCGIFADDR\_IN6 (IPv6)
- NU\_Ioctl\_SIOCGIFDSTADDR\_IN6 (IPv6)
- NU\_Ioctl\_SIOCGIFADDR (IPv4)
- NU\_Ioctl\_SIOCGIFDSTADDR (IPv4)
- NU\_Ioctl (IPv4/IPv6)

## NU\_Ioctl\_SIOCSIFHWADDR

This API sets the MAC address of a network interface by using its interface name. Note that this IOCTL is to be used when the network interface is in link-down state. You are responsible for bringing the DHCP and ARP cache into a valid state if you call this on an interface in the link-up state.

### Note



If called when the network interface is in link-up state, this function returns NU\_DEVICE\_NOT\_DOWN and will not change the MAC address.

---

## Usage

```
STATUS NU_Ioctl_SIOCSIFHWADDR(SCK_IOCTL_OPTION *option);
```

## Arguments

- option

Should contain the name of an interface whose MAC has to be fetched, the name must be assigned to option->s\_optval.

Should also contain the MAC address to be set on an interface, the MAC must be placed inside option->s\_ret.mac\_address[].

## Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
No device exists with the specified interface name.
- NU\_DEVICE\_NOT\_DOWN  
If called when the network interface is in link-up state.

## Example

```
SCK_IOCTL_OPTION option;  
UINT8 mac_address[6] = {0x00, 0x10, 0x78, 0xA3, 0x4C, 0x7F};  
  
.  
.  
.  
  
/* Populate the IOCTL structure. */  
option.s_optval = "eth0";  
memcpy(option.s_ret.mac_address, mac_address, 6);  
  
/* Bring down the interface. */  
if (NU_Ethernet_Link_Down("eth0") == NU_SUCCESS)  
{
```



```
/* Change the MAC address for this interface. */
if (NU_Ioctl_SIOCSIFHWADDR(&option) == NU_SUCCESS)
{
    printf("MAC address is updated\r\n");
}
else
{
    printf("Unable to change MAC address\r\n");
}
}

/* Bring up the interface. */
NU_Ethernet_Link_Up("eth0");
```

## Related Topics

[Net API Functions](#)

[NU\\_Ethernet\\_Link\\_Down](#)

[NU\\_Ethernet\\_Link\\_Up](#)

## NU\_Ioctl\_SIOCSPHYSADDR (IPv4)

This function sets the MAC address associated with an interface.

### Usage

```
STATUS NU_Ioctl_SIOCSPHYSADDR (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_ret.s_ipaddr` contains the IPv4 address of the device for which the MAC address is being set. The parameter `s_optval` contains the new MAC address. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE      devices[1];
SCK_IOCTL_OPTION option;
UINT8          mac_addr[6] =
                {0x00, 0x0C, 0x00, 0x0A, 0x00, 0x0B};
. . .

/* Copy the IP address of the device of which to set the MAC address. */
memcpy(option->s_ret.s_ipaddr, devices[0].dv_ip_addr,
        IP_ADDR_LEN);

/* Point to the new MAC address. */
option.s_optval = mac_addr;

/* Call NU_Ioctl_SIOCSPHYSADDR to set the MAC address. */
if (NU_Ioctl_SIOCSPHYSADDR (&option) != NU_SUCCESS)
{
    printf("Could not set the MAC address.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGIFDSTADDR\\_IN6 \(IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCSETVLAN (IPv4)

This function is responsible for setting the VLAN ID associated with the interface.

### Usage

```
STATUS NU_Ioctl_SIOCSETVLAN (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

The parameter `s_optval` contains the name of the device to modify. The parameter `s_ret.vlan_id` contains the VLAN ID to set and must be a value between 0 and 4095. The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

- NU\_INVALID\_PARM

The device specified by the option parameter is invalid.

### Example

```
DEV_DEVICE devices[1];
SCK_IOCTL_OPTION option;

. . .

/* Set the device name. */
option->s_optval = devices[0].dv_name;

/* Set the VLAN ID */
option.s_ret.vlan_id = 2;

/* Call NU_Ioctl_SIOCSETVLAN to set the VLAN ID. */
if (NU_Ioctl_SIOCSETVLAN (&option) != NU_SUCCESS)
{
    printf("Could not set the VLAN ID.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGETVLAN \(IPv4\)](#)

[NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_Ioctl\_SIOCSETVLANPRIO (IPv4/IPv6)

This function sets the VLAN priority associated with an interface. Once set, all VLAN packets transmitted from this interface will insert the specified VLAN priority into the VLAN header. This function sets the VLAN priority when using both the software and hardware methods of VLAN functionality.

### Usage

```
STATUS NU_Ioctl_SIOCSETVLANPRIO (SCK_IOCTL_OPTION *option);
```

### Arguments

- option

A pointer to the data structure that holds the name of the interface for which to set the VLAN priority and the VLAN priority value to use.

The [SCK\\_IOCTL\\_OPTION](#) data structure is defined in the [Net Data Structures](#) section. The members related to the function call are: s\_optval and vlan\_prio.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or the following Nucleus NET error code:

NU\_INVALID\_PARM

One of the input parameters is invalid.

### Example

```
SCK_IOCTL_OPTION    option;
STATUS              status;

/* Set the pointer to the interface name to configure. */
option.s_optval = "eth0";

/* Set the VLAN priority value. */
option.s_ret.vlan_prio = 64;

/* Make the function call to configure the VLAN priority value for
 * the specified interface.
 */
status = NU_Ioctl_SIOCSETVLANPRIO(&option);

if (status != NU_SUCCESS)
{
    printf("Error at call to NU_Ioctl_SIOCSETVLANPRIO.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ioctl\\_SIOCGETVLANPRIO \(IPv4/IPv6\)](#)

[NU\\_Ioctl \(IPv4/IPv6\)](#)

## NU\_IP\_Multicast\_Listen (IPv4)

This function is used to enable and disable reception of packets sent to specific IPv4 multicast addresses.

### Usage

```
STATUS NU_IP_Multicast_Listen (INT      socketd,  
                              UINT8    *interface_addr,  
                              UINT8    *multi_addr,  
                              UINT16   filter_mode,  
                              UINT8    *source_list,  
                              UINT16   num_source_addr);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **interface\_addr**  
A pointer to the IPv4 address of the interface on which reception of the specified multicast address is to be enabled or disabled. If this value is set to the unspecified address, the first non-loopback multicast-enabled interface will be used. If reception of the same multicast address is desired on more than one interface, NU\_IP\_Multicast\_Listen() is invoked separately for each desired interface.
- **multi\_addr**  
A pointer to the IPv4 multicast group to join or leave. If reception of more than one multicast address on a given interface is desired, NU\_IP\_Multicast\_Listen() is invoked separately for each desired multicast address.
- **filter\_mode**  
The filter mode for the socket. Either MULTICAST\_FILTER\_INCLUDE or MULTICAST\_FILTER\_EXCLUDE. In MULTICAST\_FILTER\_INCLUDE mode, reception of packets sent to the specified multicast address is requested only from those IPv4 source addresses listed in the source\_list parameter. In MULTICAST\_FILTER\_EXCLUDE mode, reception of packets sent to the given multicast address is requested from all IPv4 source addresses except those listed in the source\_list parameter.
- **source\_list**  
A pointer to a list of source addresses that should be included or excluded depending on the filter\_mode specified.
- **num\_source\_addr**  
The number of addresses specified in source\_list.

## Return Values

- **NU\_SUCCESS**

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- **NU\_INVALID\_PARM**

The parameter `multi_addr` is not a valid multicast address or the socket descriptor is invalid.

- **NU\_INVAL**

Insufficient memory for the new entry.

- **NU\_MEM\_ALLOC**

Insufficient memory for the new entry.

- **NU\_MULTI\_TOO\_MANY\_SRC\_ADDRS**

There are too many addresses specified in the `source_list`. Note that this parameter is bound by the macro `MAX_MULTICAST_SRC_ADDR` located at *networking/net\_cfg.h*.

## Description

This function is outlined in RFC 3376, IGMPv3. For a given combination of socket, `interface_addr`, and `multi_addr`, only a single `filter_mode` and `source_list` can be in effect at any one time. However, either the `filter_mode`, `source_list` or both, may be changed by subsequent `NU_IP_Multicast_Listen()` requests that specify the same socket, `interface_addr`, and `multi_addr`. Each subsequent request completely replaces any earlier request for the given socket, `interface_addr`, and `multi_addr`.

## Example – Join a multicast group

```
STATUS          status;
INT             socketd;
UINT8          multi_addr[] = {224,20,20,10};
NU_IOCTL_OPTION option;

/* Create the socket. */
. . .

option.s_optval = (UINT8 *)"eth0";

/* Get the IP address of the device on which we desire
 * to join the multicast group.
 */
if (NU_Ioctl_SIOCGIFADDR(&option, sizeof(option)) == NU_SUCCESS)
{
    /* Notify the IP layer to accept packets to the specified
     * multicast group from all source addresses.
     */
    status = NU_IP_Multicast_Listen(socketd,
```

```
                                option.s_ret.s_ipaddr,  
                                multi_addr, MULTICAST_FILTER_EXCLUDE,  
                                NU_NULL, 0);  
    }
```

### Example – Leave a multicast group

```
STATUS          status;  
INT             socketd;  
UINT8          multi_addr[] = {224,20,20,10};  
NU_IOCTL_OPTION option;  
  
/* Create the socket. */  
. . .  
  
/* Join a multicast group. */  
. . .  
  
option.s_optval = (UINT8 *)"eth0";  
  
/* Get the IP address of the device on which we desire  
 * to leave the multicast group.  
 */  
if (NU_Ioctl_SIOCGIFADDR(&option, sizeof(option)) == NU_SUCCESS)  
{  
    /* Notify the IP layer to reject packets to the specified  
     * multicast group.  
     */  
    status = NU_IP_Multicast_Listen(socketd,  
                                    option.s_ret.s_ipaddr,  
                                    multi_addr, MULTICAST_FILTER_INCLUDE,  
                                    NU_NULL, 0);  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_IP\\_ADD\\_MEMBERSHIP \(IPv4\)](#)

[NU\\_Setsockopt\\_IP\\_DROP\\_MEMBERSHIP \(IPv4\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)



## NU\_IP6\_Multicast\_Listen (IPv6)

This function is used to enable and disable reception of packets sent to specific IPv6 multicast addresses.

### Usage

```
STATUS NU_IP6_Multicast_Listen (INT      socketd,  
                                UINT32   device_index,  
                                UINT8     *multi_addr,  
                                UINT16    filter_mode,  
                                UINT8     *source_list,  
                                UINT16    num_source_addr);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **device\_index**  
The interface index of the IPv6-enabled interface on which reception of the specified multicast address is to be enabled or disabled. If reception of the same multicast address is desired on more than one interface, NU\_IP6\_Multicast\_Listen() is invoked separately for each desired interface.
- **multi\_addr**  
A pointer to the IPv6 multicast group to join or leave. If reception of more than one multicast address on a given interface is desired, NU\_IP6\_Multicast\_Listen() is invoked separately for each desired multicast address.
- **filter\_mode**  
The filter mode for the socket. Either MULTICAST\_FILTER\_INCLUDE or MULTICAST\_FILTER\_EXCLUDE. In MULTICAST\_FILTER\_INCLUDE mode, reception of packets sent to the specified multicast address is requested only from those IPv6 source addresses listed in the source\_list parameter. In MULTICAST\_FILTER\_EXCLUDE mode, reception of packets sent to the given multicast address is requested from all IPv6 source addresses except those listed in the source\_list parameter.
- **source\_list**  
A pointer to a list of source addresses that should be included or excluded depending on the filter\_mode specified.
- **num\_source\_addr**  
The number of addresses specified in source\_list.

## Return Values

- NU\_SUCCESS

Upon successful completion, otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- NU\_INVALID\_PARM

The parameter `multi_addr` is not a valid multicast address or the socket descriptor is invalid.

- NU\_INVAL

Insufficient memory for the new entry.

- NU\_MEM\_ALLOC

Insufficient memory for the new entry.

- NU\_MULTI\_TOO\_MANY\_SRC\_ADDRS

There are too many addresses specified in the `source_list`. Note that this parameter is bound by the macro `MAX_MULTICAST_SRC_ADDR` located at *networking/net\_cfg.h*.

## Description

This function is implemented as per the 7th revision of the Multicast Listener Discovery Version 2 (MLDv2) for IPv6 Internet Draft which updates RFC 2710. For a given combination of socket, `device_index`, and `multi_addr`, only a single `filter_mode` and `source_list` can be in effect at any one time. However, either the `filter_mode`, `source_list`, or both may be changed by subsequent `NU_IP6_Multicast_Listen()` requests that specify the same socket, `device_index`, and `multi_addr`. Each subsequent request completely replaces any earlier request for the given socket, `device_index`, and `multi_addr`.

## Example – Join a multicast group

```
STATUS          status;
INT             socketd;
UINT8          multi_addr[] =
    {0xff, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5};
UINT32         interface_index;

/* Create the socket. */
. . .

/* Get the interface index of the device on which we desire
 * to join the multicast group.
 */
interface_index =
    NU_IF_NameToIndex((UINT8 *) "eth0");

if (interface_index >= 0)
{
    /* Notify the IP layer to accept packets to the specified
```

```

    * multicast group from all source addresses.
    */
    status = NU_IP6_Multicast_Listen(socketd,
                                    interface_index,
                                    multi_addr, MULTICAST_FILTER_EXCLUDE,
                                    NU_NULL, 0);
}

```

## Example

```

/* Leave a multicast group. */

STATUS      status;
INT         socketd;
UINT8      multi_addr[] =
    {0xff, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5};
UINT32     interface_index;

/* Create the socket. */
. . .

/* Join a multicast group. */
. . .

/* Get the interface index of the device on which we desire
 * to leave the multicast group.
 */
interface_index =
    NU_IF_NameToIndex((UINT8 *) "eth0");

if (interface_index >= 0)
{
    /* Notify the IP layer to reject packets to the specified
     * multicast group.
     */
    status = NU_IP6_Multicast_Listen(socketd,
                                    interface_index,
                                    multi_addr, MULTICAST_FILTER_INCLUDE,
                                    NU_NULL, 0);
}

```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_IP\\_ADD\\_MEMBERSHIP \(IPv4\)](#)

[NU\\_Setsockopt\\_IP\\_DROP\\_MEMBERSHIP \(IPv4\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

## NU\_Is\_Connected (IPv4/IPv6)

This function is responsible for determining if a connection has been established on the specified socket descriptor. This service can only be used with sockets that are using TCP.

### Usage

```
STATUS NU_Is_Connected (INT socketd);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.

### Return Values

- `NU_TRUE`  
The socket is connected.
- `NU_FALSE`  
The socket is not connected.
- `NU_INVALID_SOCKET`  
The value specified in the `socketd` parameter is invalid.

### Example

```
INT socketd; /* A socket descriptor */  
.  
.  
.  
status = NU_Is_Connected(socketd);
```

### Related Topics

[Net API Functions](#)

[NU\\_Listen \(IPv4/IPv6\)](#)

## NU\_Is\_IPv4\_Mapped\_Addr (IPv6)

This function is responsible for determining if an IPv6 address is an IPv4-Mapped IPv6 address.

### Usage

```
INT NU_Is_IPv4_Mapped_Addr (UINT8 *addr);
```

### Arguments

- `addr`  
A pointer to the IPv6 address.

### Return Values

- `NU_TRUE`  
The address is an IPv4-Mapped IPv6 address.
- `NU_FALSE`  
The address is not an IPv4-Mapped IPv6 address.

### Example

```
UINT8 ip_addr[IP6_ADDR_LEN] =  
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0xff, 0xff, 0xc0, 0, 0, 1};  
INT status;  
  
status = NU_Is_IPv4_Mapped_Addr(ip_addr);  
.  
.  
.
```

### Related Topics

[Net API Functions](#)

[NU\\_Create\\_IPv4\\_Mapped\\_Addr \(IPv6\)](#)

## NU\_Listen (IPv4/IPv6)

This function is responsible for indicating that the server is willing to accept connection requests from clients. This function only needs to be called by the server when a connection-oriented transfer is being established.

### Usage

```
STATUS NU_Listen (INT      socketd,  
                 UINT16 backlog);
```

### Arguments

- **socketd**  
Specifies the server's socket descriptor.
- **backlog**  
Specifies the number of requests which can be queued for this server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - **NU\_NO SOCK\_MEMORY**  
There was not enough memory for Nucleus to allocate the necessary structures to be ready for a connection.

### Description

The backlog parameter specifies the number of requests that may be accepted by TCP while a higher layer is busy processing a previously accepted connection. For example, a backlog of 0 specifies that no connections will be accepted until the call to NU\_Accept is made. A backlog of 1 specifies that only one connection can be accepted while the upper-layer is processing a connection.

This routine should only be called once to set the socket to listening. However, if the routine is invoked multiple times, it will always return NU\_SUCCESS after the first successful invocation.

### Example

```
/* Backlog of 0 */  
  
INT      socketd;    /* the socket descriptor */  
INT      new_sock;
```

```

struct addr_struct peer;
.
.
.
/* With a backlog of 0, TCP will reject any incoming attempts to
 * connect until the call to NU_Accept is made.
 */
status = NU_Listen(socketd, 0);

/* Start accepting connections. */
new_sock = NU_Accept(socketd, &peer, 0);

```

## Example

```

/* Backlog of 10 */

INT      socketd;    /* the socket descriptor */
INT      new_sock;
struct addr_struct peer;
.
.
.
/* With a backlog of 10, TCP will queue up to 10 incoming
 * connections while the upper-layer is processing already
 * accepted connections.
 */
status = NU_Listen(socketd, 10);

/* Start processing connections. */
new_sock = NU_Accept(socketd, &peer, 0);

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Connect (IPv4/IPv6)</a>
<a href="#">NU_Socket (IPv4/IPv6)</a>	<a href="#">NU_Accept (IPv4/IPv6)</a>

## NU\_Ping (IPv4)

This function will transmit an IPv4 ICMP Echo Request to a foreign host and wait for a response.

### Usage

```
STATUS NU_Ping (UINT8  *dest_ip,  
               UINT32  timeout);
```

### Arguments

- `dest_ip`  
A pointer to the IPv4 address of the host to ping.
- `timeout`  
The timeout value in clock ticks. If a value of zero is supplied, the default timeout, `ICMP_DEFAULT_ECHO_TIMEOUT`, will be used.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - `NU_INVALID_PARM`  
`dest_ip` is `NU_NULL`.
  - `NU_TIMEOUT`  
A ping reply was not received within the timeout period.
  - `NU_NO_ACTION`  
Nucleus NET was unable to send the ping request. Possibly due to no route available to the foreign host.
  - `NU_NO_BUFFERS`  
The ping request could not be sent because of a lack of internal Nucleus NET buffers.
  - `NU_MEM_ALLOC`  
The ping request was sent because Nucleus NET could not allocate the needed memory to complete the service.

### Description

A timeout is used for returning from this function if the foreign host does not reply to the ping. If a timeout is not supplied, the default timeout of `ICMP_DEFAULT_ECHO_TIMEOUT` seconds as specified in *networking/net\_cfg.h* will be used. This service is useful for determining if a host on a network is up and running prior to starting some type of network communication



with them, specifically in the case of UDP where it is unknown by the protocol if the host is available.

### Example

```
UINT8 dest_addr[] = {192, 200, 100, 1};

/* Send an ICMP Echo Request to the destination address and wait
 * on the default timeout for a response.
 */
status = NU_Ping(dest_addr, 0);

if (status != NU_SUCCESS)
{
    printf("Destination address unreachable.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ping2 \(IPv4/IPv6\)](#)

## NU\_Ping2 (IPv4/IPv6)

This function is used for IPv4 and IPv6 and replaces the function NU\_Ping.

### Usage

```
STATUS NU_Ping2 (UINT8  *dest_ip,  
                UINT32  timeout,  
                INT16   family);
```

### Arguments

- **dest\_ip**  
A pointer to the address of the host to ping.
- **timeout**  
The timeout value in clock ticks. If a value of zero is supplied, the default timeout, ICMP\_DEFAULT\_ECHO\_TIMEOUT, will be used.
- **family**  
The family type of the node to ping. Either NU\_FAMILY\_IP for an IPv4 node or NU\_FAMILY\_IP6 for an IPv6 node.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_NO\_ACTION**  
NET was unable to send the ping request. Possibly due to no route being available to the foreign host.
  - **NU\_NO\_BUFFERS**  
The ping request could not be sent because of a lack of internal NET buffers.
  - **NU\_MEM\_ALLOC**  
The ping request was sent because NET could not allocate the needed memory to complete the service.
  - **NU\_INVALID\_PARM**  
One of the parameters provided is invalid.

### Description

This function will send one ICMP Echo Request to an IPv4 or IPv6 IP address and wait for an ICMP Echo Reply response. A timeout is used for returning from this function if the foreign host does not reply. If a timeout is not supplied, the default timeout of ICMP\_DEFAULT\_ECHO\_TIMEOUT seconds as specified in *networking/net\_cfg.h* will be

used. This service is useful for determining if a host on a network is up and running prior to starting some type of network communication with them, specifically in the case of UDP where it is unknown by the protocol if the host is available.

### Example

```
UINT8 dest_addr[] = {192, 200, 100, 1};

/* Send an ICMP Echo Request to the destination address and wait
 * on the default timeout for a response.
 */
status = NU_Ping2(dest_addr, 0, NU_FAMILY_IP);

if (status != NU_SUCCESS)
{
    printf("Destination address unreachable.\n");
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ping \(IPv4\)](#)

## NU\_Push (IPv4/IPv6)

This function enables or disables the value of the Nagle Algorithm. It performs the same functionality as a call to the function `NU_Setsockopt_TCP_NODELAY` or a call to the function `NU_Setsockopt` with a level of `IPPROTO_TCP` and type of `TCP_NODELAY`.

---

### Note



This function has been made obsolete in favor of the [NU\\_Setsockopt\\_TCP\\_NODELAY \(IPv4/IPv6\)](#) function.

---

## Usage

```
STATUS NU_Push (INT socketd);
```

## Arguments

- `socketd`  
Specifies a socket descriptor.

## Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_SOCKET`  
The socket parameter was an invalid socket value or it was not been previously allocated via the `NU_Socket` call.
- `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.
- `NU_INVALID`  
Invalid parameter.

## Example

```
INT socketd; /* socket descriptor */

/* Build a TCP socket.*/

. . .

/* Disable the Nagle Algorithm on the socket. */
if (NU_Push(socketd) != NU_SUCCESS)
{
    /* Fatal error, abort here. */
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_NODELAY \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

## NU\_Rarp (IPv4)

This function is responsible for dynamically resolving the IPv4 address of a diskless workstation when only the physical hardware address is known. To make use of this service, there must be a RARP server on the local network that can resolve the address.

### Usage

```
STATUS NU_Rarp (CHAR *device_name);
```

### Arguments

- `device_name`  
Pointer to the name of the device for which the IPv4 address should be resolved.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_PARM`  
A device with `device_name` could not be found.
- `NU_HOST_UNREACHABLE`  
The device with `device_name` has not been initialized. The `NU_Init_Devices` service should be used before calling `NU_Rarp`.
- `NU_NO_BUFFERS`  
There were no buffers available for building the RARP request packet.
- `NU_MEM_ALLOC`  
Indicates a memory allocation failure.
- `NU_RARP_INIT_FAILED`  
RARP failed to resolve the IP address. Possible problems are that there are no RARP servers on the local network, or the RARP server does not know the device's IP address.

### Example

```
/* An IP address of all 0's should be used when the IP address
 * is to be resolved by RARP.
 */
CHAR          ip_addr[]    = {0,0,0,0};
CHAR          subnet[]    = {255,255,255,0};
NU_DEVICE     devices[1];
NU_NET_INIT_DATA  init_struct;

/* Create the system and NET buffer memory pools. */
. . .

init_struct.net_buffered_mem = &DEMOI_NET_Memory_Pool;
init_struct.net_internal_mem = &System_Memory;
```

```
/* Initialize the network stack. */
if (NU_Init_Net(&init_struct))
{
    printf("Failed to initialize Nucleus NET.\n");
}

/* Register the MAC layer device with the stack. */
devices[0].dv_name = "NE2000_0";
devices[0].dv_hw.ether.dv_irq = 5;
devices[0].dv_hw.ether.dv_io_addr = 0x0300L;
devices[0].dv_hw.ether.dv_shared_addr = 0;
devices[0].dv_init = NE2000_Init;
devices[0].dv_flags = 0;
memcpy (devices[0].dv_ip_addr, ip_addr, 4);
memcpy (devices[0].dv_subnet_mask, subnet, 4);

NU_Init_Devices(devices, 1);

/* Now that the stack and the hardware device have been
 * initialized, NU_Rarp can be called to discover the IP
 * address for the device.
 */
if (NU_Rarp(devices[0].dv_name) != NU_SUCCESS)
{
    printf("RARP failed to discover IP address.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Push \(IPv4/IPv6\)](#)

## NU\_Recv (IPv4/IPv6)

This function is responsible for receiving data across a network during a connection-oriented transfer. Both the client and the server may call it.

### Usage

```
INT32 NU_Recv (INT      socketd,  
               CHAR     *buff,  
               UINT16  nbytes,  
               INT16   flags);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.
- **nbytes**  
Specifies the max number of bytes of data.
- **flags**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
The read side of the connection has been closed by the application, both the read and write side of the connection have been closed by the application, or the connection has been reset by the other side. The application should stop trying to receive data on this socket since successive attempts will fail. The socket may still be writable.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
- **NU\_NO\_PORT\_NUMBER**  
No local port number was stored in the socket descriptor.
- **NU\_WOULD\_BLOCK**  
Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_NO\_ROUTE\_TO\_HOST**  
This is an icmp\_error if no route to the host exists.



- **NU\_CONNECTION\_REFUSED**  
 This is an icmp\_error if the connection is refused.
- **NU\_MSG\_TOO\_LONG**  
 This is an icmp\_error if the message is too large.
- **NU\_CONNECTION\_TIMED\_OUT**  
 TCP Keep-Alive packets found that the connection has timed out.
- **NU\_SOCKET\_CLOSED**  
 The socket has been closed by another process.
- **NU\_DEST\_UNREACH\_ADMIN**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
 An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
 An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
 An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
 An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
 An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_OPTION**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_NET**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_HOST**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PROT**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_FRAG**  
 An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_SRCFAIL**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB**  
An ICMP Error Code was received on the socket.
- **NU\_SOURCE\_QUENCH**  
An ICMP Error Code was received on the socket.

### Example

```
INT socketd;          /* the original socket descriptor          */
char *buff;           /* pointer to the data buffer          */
int nbytes = 80;      /* size of the data buffer            */
int count;            /* number of bytes successfully transferred */

. . .

count = NU_Recv(socketd, buff, nbytes, 0);

/* If count contains a value greater than or equal to 0, then
   count is the number of bytes received and the data is at
   the location pointed to by buff. */
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Recv_From (IPv4/IPv6)</a>
<a href="#">NU_Recv_From_Raw (IPv4/IPv6)</a>	<a href="#">NU_Select (IPv4/IPv6)</a>
<a href="#">NU_Send (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>
<a href="#">NU_Socket (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_Getsockopt (IPv4/IPv6)</a>	

## NU\_Recvmsg (IPv4/IPv6)

This function is responsible for receiving data across a network for any socket type.

### Usage

```
INT32 NU_Recvmsg (INT    socketd,  
                  msghdr *msg,  
                  INT    flags);
```

### Arguments

- **socketd**  
Specifies a socket descriptor on which to receive data.
- **msg**  
Pointer to the [msghdr](#) data structure which contains a pointer to the buffer of data received and any ancillary data requested.
- **flags**  
This parameter is unused by Nucleus NET.

### Return Values

- The number of bytes received.  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the [NU\\_Socket](#) call.
- **NU\_NO\_PORT\_NUMBER**  
No local port number was stored in the socket descriptor.
- **NU\_INVALID\_PARM**  
One of the input parameters is invalid.
- **NU\_NOT\_CONNECTED**  
The socket is not connected.
- **NU\_WOULD\_BLOCK**  
Socket is non-blocking but blocking is required to complete the requested action.
- **NU\_NO\_DATA\_TRANSFER**  
The data transfer was not completed.
- **NU\_DEVICE\_DOWN**  
The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.

- **NU\_SOCKET\_CLOSED**  
The socket has been closed by another process.
- **NU\_NO\_ROUTE\_TO\_HOST**  
This is an ICMP error if no route to the host exists.
- **NU\_CONNECTION\_REFUSED**  
This is an ICMP error if the connection is refused.
- **NU\_MSG\_TOO\_LONG**  
This is an ICMP error if the message is too large.
- **NU\_CONNECTION\_TIMED\_OUT**  
TCP Keep-Alive packets determined that the connection has timed out.
- **NU\_DEST\_UNREACH\_ADMIN**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_OPTION**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_NET**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_HOST**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PROT**  
An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_FRAG**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_SRCFAI**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB**  
An ICMP Error Code was received on the socket.
- **NU\_SOURCE\_QUENCH**  
An ICMP Error Code was received on the socket.

## Description

The stack can also pass ancillary data to the Application Layer via this function call for UDP and RAW sockets. Note that the Application Layer must specify via a socket option call which pieces of ancillary data should be returned by the stack. This routine may be called by both the client and the server.

## Example - IPv4 using UDP - No Ancillary Data

```
INT      socketd;      /* original socket descriptor */
msghdr   msg;          /* msghdr structure */
INT32    count;        /* number of bytes transferred */
INT      flags = 0;    /* flags for the transmission */
struct   addr_struct cliaddr;
CHAR     udp_rx_buffer[UDP_SERVER_MAX_RX_SIZE];

/* Build an IPv4 UDP socket.*/
. . .

/* Bind to the socket. */
. . .

/* Set up the pointer to the client data structure. */
msg.msg_name = &cliaddr;
msg.msg_namelen = sizeof(cliaddr);

/* Set up the pointer to the buffer to use for receiving. */
msg.msg_iov = udp_rx_buffer;
msg.msg_iovlen = UDP_SERVER_MAX_RX_SIZE;

/* Set the length of the ancillary data. */
msg.msg_controllen = 0;

/* Wait for data. */
count = NU_Recvmsg (socketd, &msg, flags);

/* Check the count. */
if (count < 0)
    printf("Could not receive data.\n");
```

### Example – IPv6 using UDP – Receive Destination Address and Hop Limit as Ancillary Data

```
INT          socketd;      /* original socket descriptor */
msghdr       msg;          /* msghdr structure */
cmsghdr      *cmsg;
INT32        count;        /* number of bytes transferred */
INT          flags = 0;    /* flags for the transmission */
in6_pktinfo  pkt_info;
in6_pktinfo  *pkt_info_ptr;
INT          hop_limit;
INT          *hop_limit_ptr;
struct addr_struct cliaddr;
CHAR         udp_rx_buffer[UDP_SERVER_MAX_RX_SIZE];

/* Build an IPv6 UDP socket.*/
. . .

/* Bind to the socket. */
. . .

/* Set the socket option specifying that the stack return the
 * destination address of the packet.
 */
NU_Setsockopt_IPV6_RECVPKTINFO(socketd, 1);

/* Set the socket option specifying that the stack return the
 * hop limit of the packet.
 */
NU_Setsockopt_IPV6_HOPLIMIT(socketd, 1);

/* Set up the pointer to the client data structure. */
msg.msg_name = &cliaddr;
msg.msg_namelen = sizeof(cliaddr);

/* Set up the pointer to the buffer to use for receiving. */
msg.msg_iov = udp_rx_buffer;
msg.msg_iovlen = UDP_SERVER_MAX_RX_SIZE;

/* Set the length of the ancillary data to hold the in6_pktinfo
 * data structure and the hop limit.
 */
msg.msg_controllen =
    NU_CMSG_SPACE(sizeof(INT)) +
    NU_CMSG_SPACE(sizeof(in6_pktinfo));

/* Allocate memory for the two pieces of ancillary data. */
NU_Allocate_Memory(&System_Memory, (VOID**) &msg.msg_control,
    msg.msg_controllen, NU_NO_SUSPEND);

memset(msg.msg_control, 0, msg.msg_controllen);

/* Wait for data. */
count = NU_Recvmsg (socketd, &msg, flags);

/* Check the count. */
```

```
if (count < 0)
    printf("Could not receive data.\n");

/* Parse the ancillary data returned. */
else
{
    /* Walk through the buffer of ancillary data, searching for
     * the destination address and hop limit.
     */
    for (cmsg = NU_CMSG_FIRSTHDR(&msg);
         cmsg != NU_NULL;
         cmsg = NU_CMSG_NXTHDR(&msg, cmsg))
    {
        /* If this is the destination address structure */
        if ( (cmsg->cmsg_level == IPPROTO_IPV6) &&
             (cmsg->cmsg_type == IPV6_PKTINFO) )
        {
            /* Extract the destination address from the
             * buffer.
             */
            pkt_info_ptr = (in6_pktinfo*) CMSG_DATA(cmsg);
            pkt_info->in6_addr.family =
                pkt_info_ptr->in6_addr.family;
            memcpy(pkt_info->in6_addr.id,
                   pkt_info_ptr->in6_addr.id,
                   IPV6_ADDR_LEN);
        }

        /* If this is the hop limit structure */
        else if ( (cmsg->cmsg_level == IPPROTO_IPV6) &&
                  (cmsg->cmsg_type == IPV6_HOPLIMIT) )
        {
            /* Extract the hop limit from the buffer. */
            hop_limit_ptr = (INT*) NU_CMSG_DATA(cmsg);
            hop_limit = *hop_limit_ptr;
        }
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Send \(IPv4/IPv6\)](#)

[NU\\_Send\\_To \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

## NU\_Recv\_From (IPv4/IPv6)

This function is responsible for receiving data across a network during a connectionless transfer. It may be called by both the client and the server.

### Usage

```
INT32 NU_Recv_From (INT          socketd,  
                   CHAR          *buff,  
                   UINT16        nbytes,  
                   INT16         flags,  
                   struct addr_struct *from,  
                   INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.
- **nbytes**  
Specifies the total number of bytes of data that can be received in the buffer provided.

---

#### Note



If more than `nbytes` of data are received in a datagram, the data will be truncated after `nbytes` have been copied from the datagram.

---

- **flags**  
Used for socket compatibility. It is currently not being used.
- **from**  
Pointer to the source protocol-specific [addr\\_struct](#) structure.

---

#### Note



This structure will be filled in with the sender's side information upon successful completion of the routine.

---

- **addrlen**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion.



- **NU\_INVALID\_SOCKET**  
 The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
- **NU\_NO\_PORT\_NUMBER**  
 No local port number was stored in the socket descriptor.
- **NU\_INVALID\_PARM**  
 The addr\_struct from is NU\_NULL.
- **NU\_NOT\_CONNECTED**  
 The socket is not connected.
- **NU\_WOULD\_BLOCK**  
 Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_NO\_DATA\_TRANSFER**  
 The data transfer was not completed.
- **NU\_DEVICE\_DOWN**  
 The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.
- **NU\_SOCKET\_CLOSED**  
 The socket has been closed by another process.
- **NU\_DEST\_UNREACH\_ADMIN**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
 An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
 An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
 An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
 An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
 An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
 An ICMP Error Code was received on the socket.

- NU\_PARM\_PROB\_OPTION  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_NET  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_HOST  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_PROT  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_FRAG  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_SRCFAIL  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB  
An ICMP Error Code was received on the socket.
- NU\_SOURCE\_QUENCH  
An ICMP Error Code was received on the socket.

### Example - IPv4

```
INT socketd;      /* the original socket descriptor */
char *buff;       /* pointer to the data buffer */
struct addr_struct myaddr; /* address of server */
struct addr_struct cliaddr; /* address of client */
INT16 clilen; /* length of the client address (unused) */
INT32 count; /* number of bytes successfully transferred */
.
.
/* Build an IPv4 socket. */
.
.
/* Build the myaddr structure. */
myaddr->family = NU_FAMILY_IP; /* Specifies IPv4 */
myaddr->port = 7000; /* my server port number */

myaddr->id.is_ip_addrs[0] = 192;
myaddr->id.is_ip_addrs[1] = 9; /* my machine's */
myaddr->id.is_ip_addrs[2] = 200; /* 4-digit IP number */
myaddr->id.is_ip_addrs[3] = 1;
myaddr->name = "my_name"

/* Build the cliaddr structure. */

/* Specifies IPv4 */
cliaddr->family = NU_FAMILY_IP;

/* client's port number, will be filled in by socket library */
```

```
cliaddr->port = 0;

/* client machine's information */
cliaddr->id.is_ip_addrs[0] = 192;
cliaddr->id.is_ip_addrs[1] = 9;
cliaddr->id.is_ip_addrs[2] = 200;
cliaddr->id.is_ip_addrs[3] = 4;
cliaddr->name = "client_name"

/* Bind our address to the socket. */
NU_Bind(socketd, myaddr, 0);

/* Wait for data. */
count = NU_Recv_From (socketd, buff, strlen(buff), 0, cliaddr, &clilen);

/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");
```

### Example - IPv6

```
INT socketd;      /* the original socket descriptor */
char *buff;       /* pointer to the data buffer */
struct addr_struct myaddr; /* address of server */
struct addr_struct cliaddr; /* address of client */
INT16 clilen; /* length of the client address (unused) */
INT32 count; /* number of bytes successfully transferred */
.
.
/* Build an IPv6 socket. */
.
.
/* Build the myaddr structure. */
myaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 */
myaddr->port = 7000; /* my server port number */
myaddr->id.is_ip_addrs[0] = 0xfe;
myaddr->id.is_ip_addrs[1] = 0x80; /* my machine's */
myaddr->id.is_ip_addrs[2] = 0; /* 16-digit IP number */
myaddr->id.is_ip_addrs[3] = 0;
myaddr->id.is_ip_addrs[4] = 0;
myaddr->id.is_ip_addrs[5] = 0;
myaddr->id.is_ip_addrs[6] = 0;
myaddr->id.is_ip_addrs[7] = 0;
myaddr->id.is_ip_addrs[8] = 0;
myaddr->id.is_ip_addrs[9] = 0;
myaddr->id.is_ip_addrs[10] = 0xa;
myaddr->id.is_ip_addrs[11] = 0xb;
myaddr->id.is_ip_addrs[12] = 0xff;
myaddr->id.is_ip_addrs[13] = 0x2;
myaddr->id.is_ip_addrs[14] = 0xc;
myaddr->id.is_ip_addrs[15] = 0xd;
myaddr->name = "my_name"

/* Build the cliaddr structure. */

/* Specifies IPv6 */
cliaddr->family = NU_FAMILY_IP6;
```

```

/* client's port number, will be filled in by socket library */
cliaddr->port = 0;
/* client machine's information */
cliaddr->id.is_ip_addrs[0] = 0xfe;
cliaddr->id.is_ip_addrs[1] = 0x80;
cliaddr->id.is_ip_addrs[2] = 0;
cliaddr->id.is_ip_addrs[3] = 0;
cliaddr->id.is_ip_addrs[4] = 0;
cliaddr->id.is_ip_addrs[5] = 0;
cliaddr->id.is_ip_addrs[6] = 0;
cliaddr->id.is_ip_addrs[7] = 0;
cliaddr->id.is_ip_addrs[8] = 0;
cliaddr->id.is_ip_addrs[9] = 0;
cliaddr->id.is_ip_addrs[10] = 0xd;
cliaddr->id.is_ip_addrs[11] = 0xc;
cliaddr->id.is_ip_addrs[12] = 0xff;
cliaddr->id.is_ip_addrs[13] = 0x2;
cliaddr->id.is_ip_addrs[14] = 0xb;
cliaddr->id.is_ip_addrs[15] = 0xa;

cliaddr->name = "client_name"

/* Bind our address to the socket. */
NU_Bind(socketd, myaddr, 0);

/* Wait for data. */
count = NU_Recv_From (socketd, buff, strlen(buff), 0, cliaddr, &clilen);

/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");

```

### Example - Any IP Family

```

INT socketd; /* the original socket descriptor */
char *buff; /* pointer to the data buffer */
struct addr_struct myaddr; /* address of server */
struct addr_struct cliaddr; /* address of client */
INT16 clilen; /* length of the client address (unused) */
INT32 count; /* number of bytes successfully transferred */
.
.
/* Build an IPv4/IPv6 socket. */
.
.
/* Build the myaddr structure. */
myaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 or IPv4 */
myaddr->port = 7000; /* my server port number */
myaddr.id = IP6_ADDR_ANY;
myaddr->name = "my_name"

/* Build the cliaddr structure. */
/* Specifies IPv6 or IPv4 */
cliaddr->family = NU_FAMILY_IP6;

/* client's port number, will be filled in by socket library */
cliaddr->port = 0;
cliaddr.id = IP6_ADDR_ANY;

```

```
cliaddr->name = "client_name"

/* Bind our address to the socket. */
NU_Bind(socketd, myaddr, 0);
/* Wait for data. */
count = NU_Recv_From (socketd, buff, strlen(buff), 0, cliaddr, &clilen);

/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");
```

## Related Topics

[Net API Functions](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Send \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Recv\_From\_Raw (IPv4/IPv6)

This function is responsible for receiving data across a network during a RAW IP transfer. It may be called by both the client and the server. If multiple sockets are waiting to receive incoming RAW data of the same type, all sockets will receive a copy of the RAW data.

### Usage

```
INT32 NU_Recv_From_Raw (INT          socketd,  
                        CHAR          *buff,  
                        INT16         nbytes,  
                        INT16         flags,  
                        struct addr_struct *from,  
                        INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.

---

#### Note



Because the `nbytes` parameter is not currently used, the buffer must be large enough to hold the largest datagram that the host expects to receive. In the case of Ethernet, this can never be larger than 1500 bytes.

---

- **nbytes**  
Specifies the number of bytes of data.

---

#### Note



This parameter is not currently used. Instead the last datagram received is copied as a whole into the location pointed to by the `buff` parameter.

---

- **flags**  
This parameter is currently unused.
- **from**  
Pointer to the source protocol-specific [addr\\_struct](#) structure.
- **addrlen**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- **NU\_INVALID\_SOCKET**  
 The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
- **NU\_NO\_DATA\_TRANSFER**  
 The data transfer was not completed.
- **NU\_INVALID\_PARM**  
 The addr\_struct from is NU\_NULL.
- **NU\_NOT\_CONNECTED**  
 The socket is not connected.
- **NU\_WOULD\_BLOCK**  
 Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_NO\_PORT\_NUMBER**  
 No port number.
- **NU\_DEVICE\_DOWN**  
 The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.

### Example - IPv4

```

INT socketd;      /* the original socket descriptor */
char *buff;       /* pointer to the data buffer */
struct addr_struct cliaddr; /* address we are receiving from */
int cliilen; /* length of the client address (unused) */

/* number of bytes successfully transferred */
int count;
.
.
/* Build an IPv4 socket. */
.
.
/* Build the cliaddr structure. */
cliaddr->family = NU_FAMILY_IP; /* Specifies IPv4 */
cliaddr->port = 0;
cliaddr->id.is_ip_addrs[0] = 192;
cliaddr->id.is_ip_addrs[1] = 9; /* client machine's */
cliaddr->id.is_ip_addrs[2] = 200; /* 4-digit IP number */
cliaddr->id.is_ip_addrs[3] = 4;
cliaddr->name = "client_name"

/* Wait for data from the IPv4 client. */
count = NU_Recv_From_Raw (socketd, buff, strlen(buff), 0, cliaddr,
                        cliilen);

```

```
/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");
```

### Example - IPv6

```
INT socketd;    /* the original socket descriptor */
char *buff;     /* pointer to the data buffer */
struct addr_struct cliaddr; /* address from which you are receiving */
int clilen;     /* length of the client address (unused) */

/* number of bytes successfully transferred */
int count;
.
.
/* Build an IPv6 socket. */
.
.
/* Build the cliaddr structure. */
cliaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 */
cliaddr->port = 0;
cliaddr->id.is_ip_addrs[0] = 0xfe;
cliaddr->id.is_ip_addrs[1] = 0x80; /* client machine's */
cliaddr->id.is_ip_addrs[2] = 0; /* 16-digit IP number */
cliaddr->id.is_ip_addrs[3] = 0;
cliaddr->id.is_ip_addrs[4] = 0;
cliaddr->id.is_ip_addrs[5] = 0;
cliaddr->id.is_ip_addrs[6] = 0;
cliaddr->id.is_ip_addrs[7] = 0;
cliaddr->id.is_ip_addrs[8] = 0;
cliaddr->id.is_ip_addrs[9] = 0;
cliaddr->id.is_ip_addrs[10] = 0xa;
cliaddr->id.is_ip_addrs[11] = 0xb;
cliaddr->id.is_ip_addrs[12] = 0xff;
cliaddr->id.is_ip_addrs[13] = 0x02;
cliaddr->id.is_ip_addrs[14] = 0xc;
cliaddr->id.is_ip_addrs[15] = 0xd;

cliaddr->name = "client_name"

/* Wait for data from the IPv6 client. */
count = NU_Recv_From_Raw (socketd, buff, strlen(buff), 0, cliaddr,
                          clilen);

/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");
```

### Example - Any IP Family

```
INT socketd;    /* the original socket descriptor */
char *buff;     /* pointer to the data buffer */
struct addr_struct cliaddr; /* address from which we are receiving */
int clilen;     /* length of the client address (unused) */

/* number of bytes successfully transferred */
int count;
```



```
.
.
/* Build an IPv4/IPv6 socket. */
.
.
/* Build the cliaddr structure. */
cliaddr->family = NU_FAMILY_IP6;
cliaddr->port = 0;
cliaddr->id = IP6_ADDR_ANY;
cliaddr->name = "client_name"

/* Wait for data from any client. */
count = NU_Recv_From_Raw (socketd, buff, strlen(buff), 0, cliaddr,
                          clilen);

/* Check for errors. */
if (count < 0)
    printf("Could not receive data.\n");
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Recv (IPv4/IPv6)</a>
<a href="#">NU_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_Select (IPv4/IPv6)</a>
<a href="#">NU_Send (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>
<a href="#">NU_Socket (IPv4/IPv6)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Recv\_IF\_Addr (IPv4)

This service is used to obtain the IPv4 address of the interface that received the last incoming datagram.

### Usage

```
STATUS NU_Recv_IF_Addr (INT    socketd,  
                       UINT8  *if_addr);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `if_addr`  
Pointer to a 4-byte array used to return the IPv4 address of the interface.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_SOCKET`  
The socket descriptor is not valid.
- `NU_INVALID_PARM`  
The `if_addr` pointer is not valid.
- `NU_NOT_CONNECTED`  
The socket is not connected.

### Description

To use this function, you must first enable the socket option `IP_RECVIFADDR` using `NU_Setsockopt()` or `NU_Setsockopt_IP_RECIFADDR()`. Once enabled, this function can be invoked after a call to `NU_Recv_From()`. The interface address of the most recently received packet is returned. This service is useful for multi-interface enabled devices which receive multicast or broadcast packets. Note that this service is only valid for UDP sockets.

### Example

```
INT    socketd; /* the socket descriptor */  
STATUS status;  
UINT8  if_ip[IP_ADDR_LEN];  
  
. . .  
  
/* Read the IP address of the interface that received the last  
 * datagram.  
 */  
status = NU_Recv_IF_Addr(socketd, if_ip);
```

```
/* Check for errors. */  
if (status != NU_SUCCESS)  
    printf("Could not get interface IP address.\n");
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_IP\\_RECVIFADDR \(IPv4\)](#)

[NU\\_Getsockopt\\_IP\\_RECVIFADDR \(IPv4\)](#)

## NU\_Remove\_Device (IPv4/IPv6)

This function removes an interface from the system.

### Usage

```
STATUS NU_Remove_Device (CHAR    *name,  
                        UINT32  flags);
```

### Arguments

- **name**  
Pointer to the name of the interface to remove.
- **flags**  
Flags indicating special conditions for the deletion. There are currently no flags defined. This parameter is for future use.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
There is no interface in the system with the associated name.

### Description

All resources allocated for the interface are removed, all routes using the interface are deleted, and the driver is notified to release all resources allocated.

If the IPv4 address for the interface was obtained via DHCP, you should call the routine `NU_DHCP_Release()` before calling `NU_Remove_Device()` to gracefully release the address with the DHCP server.

#### Note



Some drivers may not support the ioctl request to remove the device. The implementor may need to add support to the driver's ioctl routine to process the `DEV_REMDEV` command.

---

### Example

```
/* Initialize an interface. */  
. . .  
  
/* Remove the the interface from the system */  
if (NU_Remove_Device("Ethernet_0", 0) != NU_SUCCESS)  
{  
    /* Log an error. */  
    printf("Could not remove device.\n");  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Init\\_Devices \(IPv4/IPv6\)](#)

[NU\\_Attach\\_IP\\_To\\_Device \(IPv4\)](#)

[NU\\_Detach\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)

[NU\\_Remove\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)

[NU\\_Add\\_IP\\_To\\_Device \(IPv6\)](#)

## NU\_Remove\_IP\_From\_Device (IPv4/IPv6)

This function removes an IP address from an interface.

### Usage

```
STATUS NU_Remove_IP_From_Device (CHAR    *name,  
                                UINT8    *ip_addr,  
                                INT16    family);
```

### Arguments

- **name**  
Pointer to the name of the interface.
- **ip\_addr**  
A pointer to the IP address to remove from the interface.
- **family**  
The family type of the IP address to remove; either `NU_FAMILY_IP` for IPv4 or `NU_FAMILY_IP6` for IPv6.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
No interface exists with the specified name, or the IP address does not exist on the interface.
- **NU\_INVALID\_ADDRESS**  
The IP address was obtained via DHCP and cannot be removed using this API routine. You must use `NU_Dhcp_Release` to remove an IP address obtained via DHCP.

### Description

All routes using the IP address as a next-hop are deleted, all client sockets bound to the IP address are invalidated, and any tasks suspended on the sockets are resumed. If the specified address is the only address on the interface, this routine also disables the interface from transmitting data.

If you delete an IPv4 link-local address using this interface, IPv4 link-local address configuration will be disabled on the interface. If you delete all globally routable addresses using this interface, and IPv4 link-local configuration has been previously enabled on the interface, IPv4 link-local address configuration will be initiated in an effort to obtain an IPv4 link-local address for the interface.

### Example

```
UINT8 ip_addr[4] = {192, 168, 10, 210};  
  
/* Initialize the interface with IP address ip_addr. */
```

```
/* Remove the IP address from the interface. */
if (NU_Remove_IP_From_Device("Ethernet_0", ip_addr,
                             NU_FAMILY_IP) != NU_SUCCESS)
{
    /* Log an error. */
    printf("Could not remove IP address from device.\n");
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Attach\\_IP\\_To\\_Device \(IPv4\)](#)

[NU\\_Detach\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)

[NU\\_Remove\\_Device \(IPv4/IPv6\)](#)

[NU\\_Add\\_IP\\_To\\_Device \(IPv6\)](#)

## NU\_Rip2\_Initialize (IPv4)

This service is called to initialize the IPv4 RIP2 module.

### Usage

```
STATUS NU_Rip2_Initialize (RIP2_STRUCT *ri_ptr,  
                           INT          num);
```

### Arguments

- **ri\_ptr**  
A pointer to an array of RIP2 initialization structures. There must be one element in this array for each network interface that should be used with RIP2. The [RIP2\\_STRUCT](#) data structure is defined in the [Net Data Structures](#) section.
- **num**  
The number of items in the array pointed to by **ri\_ptr**.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
An input parameter is invalid. Note that multicasting must be included if the Send or Receive mode is set to RIPv2 only.
- **NU\_MEM\_ALLOC**  
A memory allocation failed.
- **NU\_INVALID**  
A general purpose error condition. Probably indicates that a required resource, task, semaphore, and so on could not be created.

### Description

RIP2 executes as an independent task in the system that periodically updates the IPv4 Routing Table. Once initialized, no further interaction with RIP2 is required.

### Example

```
VOID RIP2_exec_task(UNSIGNED argc, VOID *argv)
{
    STATUS      status;
    DEV_DEVICE  devices[1];
    RIP2_STRUCT rip2[1];

    /* Initialize Nucleus NET and each device. */
    . . .

    /* Set up the init structure to send and receive only RIP2
     * packets on this interface. All RIP1 packets will be
```



```
    * ignored by this interface.
    */
    rip2[0].rip2_device_name = devices[0].dv_name;
    rip2[0].rip2_metric = 1;
    rip2[0].rip2_sendmode = SEND_RIP2;
    rip2[0].rip2_recvmode = RECV_RIP2;

    /* Call the init function to invoke RIP2 on one device. */
    status = NU_Rip2_Initialize (rip2, 1);

    /* Check the status of the operation. */
    if (status != NU_SUCCESS)
    {
        printf("Failed to initialize RIP.\n");
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Ripng\\_Initialize \(IPv6\)](#)

## NU\_Ripng\_Initialize (IPv6)

This service is called to initialize the IPv6 RIPng module.

### Usage

```
STATUS NU_Ripng_Initialize (const RIPNG_STRUCT *ripng,  
                           INT num);
```

### Arguments

- **ripng**  
A pointer to an array of RIPng initialization structures. There must be one element in this array for each IPv6-enabled network interface that will send and receive RIPng packets. The [RIPNG\\_STRUCT](#) data structure is defined in the [Net Data Structures](#) section.
- **num**  
The number of items in the array pointed to by ripng.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
Multicasting is not enabled in the system or none of the interfaces passed into the routine could be initialized for RIPng.
- **NU\_MEM\_ALLOC**  
A memory allocation failed.
- **NU\_INVALID**  
A general purpose error condition. It probably indicates that a required resource, task, semaphore, and so on could not be created.

### Description

RIPng executes as an independent task in the system that periodically updates the IPv6 Routing Table. Once initialized, no further interaction with RIPng is required from the application.

Note that multicasting must be enabled in the system and on each interface that will perform RIPng. RIPng uses multicast packets to propagate routing information.

RIPng version 1 is specified in RFC 2080.

### Example

```
VOID RIPng_exec_task(UNSIGNED argc, VOID *argv)  
{  
    STATUS      status;  
    DEV_DEVICE  devices[1];  
    RIPNG_STRUCT ripng[1];
```

```
/* Initialize Nucleus NET. */
. . .

/* Initialize each of the devices in the system. */
. . .

/* Initialize RIPng for one device .*/

/* Setup the init structure. */
ripng[0].ripng_device_name = devices[0].dv_name;
ripng[0].ripng_metric      = 1;

/* Call the init function to invoke RIPng on one device. */
status = NU_Ripng_Initialize (ripng, 1);

/* Check the status of the operation. */
if (status != NU_SUCCESS)
{
    printf("Failed to initialize RIPng.\n");
}
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Rip2\\_Initialize \(IPv4\)](#)

## NU\_Select (IPv4/IPv6)

The NU\_Select service examines multiple sockets to determine if any of the sockets are ready for reading or writing.

### Usage

```
STATUS NU_Select (INT      max_sockets,  
                  FD_SET  *readfs,  
                  FD_SET  *writefs,  
                  FD_SET  *exceptfs,  
                  UNSIGNED timeout);
```

### Arguments

- **max\_sockets**  
Specifies the value of the largest socket descriptor that should be checked plus one. For example, if a select on socket descriptors 1, 6, and 8 is desired, then the value of max\_sockets should be 9 (8 + 1).
- **readfs**  
Pointer to the bitmap of sockets on which the caller wants to check for data. This value can be set to NU\_NULL if the caller does not want to check any sockets for data.
- **writefs**  
Pointer to the bitmap of sockets which the caller wants to check for writability. This value can be set to NU\_NULL if the caller does not want to check any sockets for writability. A socket is writable if the TCP socket is connected and there is room in the window to write more data, buffers are available to write to the UDP or RAW socket, there is an ICMP error waiting on the socket, or the write-half of the connection has been closed.
- **exceptfs**  
This parameter is not currently used. The caller should pass in NU\_NULL.
- **timeout**  
Specifies how long, in ticks, to suspend if there is no data on any of the sockets.

The following options are available for the timeout parameter:

NU\_NO\_SUSPEND

The service returns immediately regardless of whether the request can be satisfied.

NU\_SUSPEND

The calling task is suspended until at least one of the specified sockets contains data.

timeout value (1 - 4,294,967,293)

The calling task is suspended until data is available or until the specified number of timer ticks have expired.

## Return Values

- **NU\_SUCCESS**  
Upon successful completion, at least one of the specified sockets will be data ready.
- **NU\_NO\_SOCKETS**  
The readfs and writefs parameters are both NU\_NULL or there were no bits set in either input bitmap.
- **NU\_INVALID\_SOCKET**  
No valid sockets were specified in the readfs parameter.
- **NU\_NO\_DATA**  
Indicates whether or not there are any other possible return values.
- **-1**  
NU\_Select failed generally.
- **NU\_DEST\_UNREACH\_ADMIN**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_OPTION**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_NET**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_HOST**  
An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_PROT  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_FRAG  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_SRCFAIL  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB  
An ICMP Error Code was received on the socket.
- NU\_SOURCE\_QUENCH  
An ICMP Error Code was received on the socket.

## Description

Bitmaps are used to indicate which sockets should be checked, each bit corresponding to a specific socket. This service is useful when a task wants to suspend pending an event on multiple sockets, or when a task needs to timeout if the event does not occur by a certain time. The NU\_Recv(), NU\_Recv\_From(), and NU\_Recvmsg() calls do not allow the caller to specify a timeout value. The caller must choose whether to suspend indefinitely.

The NU\_Select service can be used to check for data ready sockets, acceptable connections, or sockets which are ready for writing. Both UDP and TCP client and server sockets can be checked for received data and writability.

TCP servers can use NU\_Select to check multiple sockets for acceptable connections. The readfs parameter is also used to indicate those sockets to be checked for acceptable connections.

## Example

```
int i;
FD_SET readfs;
INT sock1, sock2, sock3;
INT max_socket;

/* Create three sockets. */
.
.
.

/* Make sure all bits are initially set to 0. */
NU_FD_Init(&readfs);

/* Set the read bit for the three sockets. */
NU_FD_Set(sock1, &readfs);
NU_FD_Set(sock2, &readfs);
NU_FD_Set(sock3, &readfs);

/* Determine which socket is the highest numbered. */
if ( (sock1 > sock2) && (sock1 > sock3) )
```

```
    max_socket = sock1;

else if (sock2 > sock3)
    max_socket = sock2;

else
    max_socket = sock3;

/* Wait five seconds for data to arrive on one of the sockets. */
if (NU_Select(max_socket + 1, &readfs, NU_NULL,
             NU_NULL, 5) == NU_SUCCESS)
{
    /* At least one of the sockets received data.
     * Check all of them to find those that have data.
     */
    for (i=0; i <= max_socket; i++)
    {
        if (NU_FD_Check(i, &readfs) == NU_FALSE)
            continue;
        .
        .
        .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_FD\\_Init \(IPv4/IPv6\)](#)

[NU\\_FD\\_Reset \(IPv4/IPv6\)](#)

[NU\\_FD\\_Set \(IPv4/IPv6\)](#)

[NU\\_FD\\_Check \(IPv4/IPv6\)](#)

## NU\_Send (IPv4/IPv6)

This function is responsible for transmitting data across a network during a connection-oriented transfer. It may be called by both the client and the server.

### Usage

```
INT32 NU_Send (INT      socketd,  
               CHAR     *buff,  
               UINT16   nbytes,  
               INT16    flags);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.
- **nbytes**  
Specifies the number of bytes of data. Valid values for this parameter are in the range 0-IP\_MAX\_DATA\_SIZE (65,495).
- **flags**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - NU\_INVALID\_SOCKET  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
  - NU\_NO\_PORT\_NUMBER  
No local port number was stored in the socket descriptor.
  - NU\_NOT\_CONNECTED  
The data transfer was not completed. This probably occurred because the connection was closed for some reason.
  - NU\_NO\_ROUTE\_TO\_HOST  
This is an icmp\_error if no route to the host exists.
  - NU\_CONNECTION\_REFUSED



This is an icmp\_error if the connection is refused.

- **NU\_MSG\_TOO\_LONG**

This is an icmp\_error if the message is too large.

- **NU\_WOULD\_BLOCK**

Socket is non-blocking, but blocking is required to complete the requested action.

- **NU\_INVALID\_PARAM**

The buffer of data to send is NULL, and the number of bytes is not zero.

- **NU\_CONNECTION\_TIMED\_OUT**

TCP Keep-Alive packets found that the connection has timed out.

- **NU\_SOCKET\_CLOSED**

The socket has been closed by another process.

- **NU\_DEST\_UNREACH\_ADMIN**

An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_ADDRESS**

An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_PORT**

An ICMP Error Code was received on the socket.

- **NU\_TIME\_EXCEED\_HOPLIMIT**

An ICMP Error Code was received on the socket.

- **NU\_TIME\_EXCEED\_REASM**

An ICMP Error Code was received on the socket.

- **NU\_PARM\_PROB\_HEADER**

An ICMP Error Code was received on the socket.

- **NU\_PARM\_PROB\_NEXT\_HDR**

An ICMP Error Code was received on the socket.

- **NU\_PARM\_PROB\_OPTION**

An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_NET**

An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_HOST**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PROT**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_FRAG**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_SRCFAIL**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB**  
An ICMP Error Code was received on the socket.
- **NU\_SOURCE\_QUENCH**  
An ICMP Error Code was received on the socket.

### Example

```
INT socketd;      /* the original socket descriptor */
char*buff = "x"; /* pointer to the data buffer */
int nbytes = 1;   /* number of bytes to be sent */
int count;        /* number of bytes successfully transferred/
.
.
.
count = NU_Send(socketd, buff, nbytes, 0);

/* If count contains a value greater than or equal to 0, then count is the
number of bytes transferred. */
.
.
.
```

### Related Topics

[Net API Functions](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

[NU\\_Send\\_To \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send \(IPv4/IPv6\)](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

## NU\_Sendmsg (IPv4/IPv6)

This function is responsible for transmitting data across a network for any socket type.

### Usage

```
INT32 NU_Sendmsg (INT    socketd,  
                  msghdr *msg,  
                  INT    flags);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **msg**  
Pointer to the [msghdr](#) data structure which contains a pointer to the buffer of data to transmit and any ancillary data specified.
- **flags**  
This parameter is currently unused.

### Return Values

- The number of bytes transmitted.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - **NU\_NO\_PORT\_NUMBER**  
No local port number was stored in the socket descriptor.
  - **NU\_NOT\_CONNECTED**  
The data transfer was not completed. This probably occurred because the connection was closed for some reason.
  - **NU\_NO\_ROUTE\_TO\_HOST**  
This is an ICMP error if no route to the host exists.
  - **NU\_CONNECTION\_REFUSED**  
This is an ICMP error if the connection is refused.
  - **NU\_MSG\_TOO\_LONG**  
This is an ICMP error if the message is too large.

- **NU\_WOULD\_BLOCK**  
Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_INVALID\_PARAM**  
The buffer of data to send is NULL, and the number of bytes is not zero.
- **NU\_CONNECTION\_TIMED\_OUT**  
TCP Keep-Alive packets found that the connection has timed out.
- **NU\_NO\_DATA\_TRANSFER**  
The data transfer was not completed.
- **NU\_INVALID\_ADDRESS**  
The address passed in was most likely incomplete (that is, missing the IP or port number).
- **NU\_DEVICE\_DOWN**  
The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.
- **NU\_SOCKET\_CLOSED**  
The socket has been closed by another process.
- **NU\_DEST\_UNREACH\_ADMIN**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**

- An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_OPTION  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_NET  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_HOST  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_PROT  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_FRAG  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_SRCFAIL  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB  
An ICMP Error Code was received on the socket.
- NU\_SOURCE\_QUENCH  
An ICMP Error Code was received on the socket.

## Description

The Application Layer can also pass ancillary data to the stack via this function call for UDP and IPRAW sockets. This routine may be called by both the client and the server.

## Example

```
/* IPv4 using UDP - No Ancillary Data */

INT      socketd;      /* original socket descriptor */
CHAR      *buff = "x"; /* pointer to the data buffer */
msghdr    msg;         /* msghdr structure */
INT32     count;       /* number of bytes transferred */
INT       flags = 0;    /* flags for the transmission */

/* Build an IPv4 UDP socket.*/
. . .

/* Build the server's address structure. */
msg.msg_name.servaddr->family = NU_FAMILY_IP; msg.msg_name.servaddr->port
= 0;

msg.msg_name.servaddr->id.is_ip_addrs[0] = 192;
```

---

```

msg.msg_name.servaddr->id.is_ip_addrs[1] = 9; msg.msg_name.servaddr-
id.is_ip_addrs[2] = 200; msg.msg_name.servaddr->id.is_ip_addrs[3] = 1;

msg.msg_name.servaddr->name = "client_name"

msg.msg_buf = buff;
msg.nbytes = strlen(buff);

/* Send the data to the IPv4 node. */
count = NU_Sendmsg (socketd, &msg, flags);

/* Check the count. */
if (count < 0)
    printf("bad error occurred.\n\r");

```

## Example

```

/* IPv6 using UDP - Pass Source Address and Hop Limit as Ancillary Data*/

INT      socketd;      /* original socket descriptor */
CHAR      *buff = "x"; /* pointer to the data buffer */
msghdr    msg;          /* msghdr structure */
cmsghdr   *cmsg;
INT32     count;        /* number of bytes transferred */
INT       flags = 0;    /* flags for the transmission */
CHAR      *anc_buff;    /* Buffer for ancillary data */
in6_pktinfo *pkt_info;
INT       *hop_limit;
INT       length;

/* Build an IPv6 UDP socket.*/
. . .

/* Allocate memory for the two pieces of ancillary data. */
length = NU_CMSG_SPACE(sizeof(in6_pktinfo)) + NU_CMSG_SPACE(sizeof(INT));

NU_Allocate_Memory(&System_Memory, (VOID**)&anc_buff, length,
                  NU_NO_SUSPEND);

/* Set up the msghdr to point to the buffer of ancillary data. */
msg.msg_control = anc_buff;
msg.msg_controllen = length;

/* Get a pointer to the first cmsghdr in the buffer. */
cmsg = NU_CMSG_FIRSTHDR(&msg);

/* Set the level and type for specifying the source address */
cmsg->cmsg_level = IPPROTO_IPV6;
cmsg->cmsg_type = IPV6_PKTINFO;

/* Set the length of the source address object. */
cmsg->cmsg_len = NU_CMSG_LEN(sizeof(in6_pktinfo));

/* Fill in the source address. */
pkt_info = (in6_pktinfo*)NU_CMSG_DATA(cmsg);
memcpy(pkt_info->ipi6_addr, source_address, IP6_ADDR_LEN);

/* Let the stack select the interface out of which to transmit the

```

```

    * packet.
    */
    pkt_info->ipi6_ifindex = 0;

    /* Get a pointer to the next cmsghdr structure and fill in the
    * hop limit.
    */

    cmsg = NU_CMSG_NXTHDR(&msg, cmsg);

    /* Set the level and type for specifying the hop limit. */
    cmsg->cmsg_level = IPPROTO_IPV6;
    cmsg->cmsg_type = IPV6_HOPLIMIT;

    /* Set the length of the hop limit object. */
    cmsg->cmsg_len = NU_CMSG_LEN(sizeof(INT));

    /* Fill in the hop limit. */
    hop_limit = (INT*)NU_CMSG_DATA(cmsg);
    *hop_limit = 64;

    /* Build the server's address structure. */
    msg.msg_name.servaddr->family = NU_FAMILY_IP6; msg.msg_name.servaddr->port = 0;

    msg.msg_name.servaddr->id.is_ip_addrs[0] = 0xfe;
    msg.msg_name.servaddr->id.is_ip_addrs[1] = 0x80;
    msg.msg_name.servaddr->id.is_ip_addrs[2] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[3] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[4] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[5] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[6] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[7] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[8] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[9] = 0;
    msg.msg_name.servaddr->id.is_ip_addrs[10] = 0xa;
    msg.msg_name.servaddr->id.is_ip_addrs[11] = 0xb;
    msg.msg_name.servaddr->id.is_ip_addrs[12] = 0xff;
    msg.msg_name.servaddr->id.is_ip_addrs[13] = 0x02;
    msg.msg_name.servaddr->id.is_ip_addrs[14] = 0xc;
    msg.msg_name.servaddr->id.is_ip_addrs[15] = 0xd;

    msg.msg_name.servaddr->name = "client_name"

    msg.msg_buf = buff;
    msg.nbytes = strlen(buff);

    /* Send the data to the IPv4 node. */
    count = NU_Sendmsg (socketd, &msg, flags);

    /* Check the count. */
    if (count < 0)
        printf("bad error occurred.\n\r");

```

## Related Topics

[Net API Functions](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

[NU\\_Send\\_To \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Send \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)



## NU\_Send\_To (IPv4/IPv6)

This function is used by UDP applications to transmit data across a network during a connectionless transfer.

### Usage

```
INT32 NU_Send_To (INT          socketd,  
                  CHAR          *buff,  
                  UINT16        nbytes,  
                  INT16         flags,  
                  struct addr_struct *to,  
                  INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.
- **nbytes**  
Specifies the number of bytes of data.
- **flags**  
This parameter is currently unused.
- **to**  
Pointer to the destination's protocol-specific [addr\\_struct](#) structure.
- **addrlen**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - **NU\_INVALID\_PARM**  
The [addr\\_struct](#) is `NU_NULL`.
  - **NU\_NOT\_CONNECTED**  
If the socket is not connected.

- **NU\_NO\_PORT\_NUMBER**  
No local port number was stored in the socket descriptor.
- **NU\_NO\_DATA\_TRANSFER**  
The data transfer was not completed.
- **NU\_INVALID\_ADDRESS**  
The address passed in was most likely incomplete (for example, missing the IP or port number).
- **NU\_WOULD\_BLOCK**  
Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_DEVICE\_DOWN**  
The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.
- **NU\_SOCKET\_CLOSED**  
The socket has been closed by another process.
- **NU\_DEST\_UNREACH\_ADMIN**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_ADDRESS**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PORT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_HOPLIMIT**  
An ICMP Error Code was received on the socket.
- **NU\_TIME\_EXCEED\_REASM**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_HEADER**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_NEXT\_HDR**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB\_OPTION**

An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_NET

An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_HOST

An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_PROT

An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_FRAG

An ICMP Error Code was received on the socket.

- NU\_DEST\_UNREACH\_SRCFAIL

An ICMP Error Code was received on the socket.

- NU\_PARM\_PROB

An ICMP Error Code was received on the socket.

- NU\_SOURCE\_QUENCH

An ICMP Error Code was received on the socket.

## Description

Both UDP clients and UDP servers may call `NU_Send_To`. The largest block of data that can be transmitted is 65,507 bytes. There are 28 bytes of overhead for the IP and UDP headers.

## Example - IPv4

```
INT socketd; /* the original socket descriptor */
char *buff = "x"; /* pointer to the data buffer */
struct addr_struct servaddr; /* address of the remote host */
int servlen; /* length of the server address (unused) */
int count; /* number of bytes successfully transferred */
.
.
.
/* Build a socket.*/
.
.
.
/* Build the server's address structure. */

servaddr->family = NU_FAMILY_IP; /* Specifies IPv4 */
servaddr->port = 6000; /* server's port number */

servaddr->id.is_ip_addrs[0] = 192;
servaddr->id.is_ip_addrs[1] = 9; /* the server machine's */
servaddr->id.is_ip_addrs[2] = 200; /* 4-digit IP number */
```

```

servaddr->id.is_ip_addrs[3] = 1;
servaddr->name = "server_name"

/* Send the data to the IPv4 node. */
count = NU_Send_To (socketd, buff, strlen(buff), 0, servaddr,
                    servlen);

/* Check the count. */
if (count < 0)
    print("bad error occurred.\n\r");

```

### Example - IPv6

```

INT socketd; /* the original socket descriptor */
char *buff = "x"; /* pointer to the data buffer */
struct addr_struct servaddr; /* address of the remote host */
int servlen; /* length of the server address (unused) */
int count; /* number of bytes successfully transferred */
.
.
.
/* Build a socket.*/
.
.
.
/* Build the server's address structure. */

servaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 */
servaddr->port = 6000; /* server's port number */

servaddr->id.is_ip_addrs[0] = 0xfe;
servaddr->id.is_ip_addrs[1] = 0x80; /* the server machine's */
servaddr->id.is_ip_addrs[2] = 0; /* 16-digit IP number */
servaddr->id.is_ip_addrs[3] = 0;
servaddr->id.is_ip_addrs[4] = 0;
servaddr->id.is_ip_addrs[5] = 0;
servaddr->id.is_ip_addrs[6] = 0;
servaddr->id.is_ip_addrs[7] = 0;
servaddr->id.is_ip_addrs[8] = 0;
servaddr->id.is_ip_addrs[9] = 0;
servaddr->id.is_ip_addrs[10] = 0xa;
servaddr->id.is_ip_addrs[11] = 0xb;
servaddr->id.is_ip_addrs[12] = 0xff;
servaddr->id.is_ip_addrs[13] = 0x02;
servaddr->id.is_ip_addrs[14] = 0xc;
servaddr->id.is_ip_addrs[15] = 0xd;

servaddr->name = "server_name"

/* Send the data to the IPv6 node. */
count = NU_Send_To (socketd, buff, strlen(buff), 0, servaddr, servlen);

/* Check the count. */
if (count < 0)
    print("bad error occurred.\n\r");

```

## Related Topics

[Net API Functions](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Select \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Socket \(IPv4/IPv6\)](#)

[NU\\_CMSG\\_DATA \(IPv6\)](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_Send \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## NU\_Send\_To\_Raw (IPv4/IPv6)

This function is responsible for transmitting data across a network during a RAW IP transfer. It may be called by both the client and the server.

### Usage

```
INT32 NU_Send_To_Raw (INT          socketd,  
                     CHAR          *buff,  
                     UINT16        nbytes,  
                     INT16         flags,  
                     struct addr_struct *to,  
                     INT16         addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **buff**  
Pointer to the data buffer.
- **nbytes**  
Specifies the number of bytes of data.
- **flags**  
This parameter is currently unused.
- **to**  
Pointer to the destination's protocol-specific [addr\\_struct](#) structure. Since the TCP/IP stack does not build a transport layer header for RAW IP packets, the port member of this parameter must always be set to zero by the application.
- **addrlen**  
This parameter is currently unused.

### Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
  - **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - **NU\_NO\_DATA\_TRANSFER**  
The data transfer was not completed.
  - **NU\_INVALID\_PARM**

The `addr_struct` to is `NU_NULL`.

- `NU_NOT_CONNECTED`

The socket is not connected.

- `NU_WOULD_BLOCK`

Socket is non-blocking, but blocking is required to complete the requested action.

- `NU_INVALID_ADDRESS`

The address passed in was most likely incomplete (for example, missing the IP number).

- `NU_NO_PORT_NUMBER`

The port number does not exist.

- `NU_DEVICE_DOWN`

The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.

### Example - IPv4

```
INT socketd;                /* original socket descriptor */
char *buff = "x";          /* pointer to the data buffer */
struct addr_struct servaddr; /* address of the remote host */
int servlen;               /* Unused */
int count;                 /* number of bytes transferred */

/* Build a socket.*/

/* Build the server's address structure. */
servaddr->family = NU_FAMILY_IP; /* Specifies IPv4 */
servaddr->port = 0;              /* server's port number */

servaddr->id.is_ip_addrs[0] = 192;
servaddr->id.is_ip_addrs[1] = 9; /* the server machine's */
servaddr->id.is_ip_addrs[2] = 200; /* 4-digit IP number */
servaddr->id.is_ip_addrs[3] = 1;
servaddr->name = "server_name"

/* Send the data to the IPv4 node. */
count = NU_Send_To_Raw (socketd, buff, strlen(buff), 0,
                       servaddr, servlen);

/* Check the count. */
if (count < 0)
    printf("bad error occurred.\n\r");
```

### Example - IPv6

```
INT socketd;                /* original socket descriptor */
char *buff = "x";          /* pointer to the data buffer */
```

```

struct addr_struct servaddr;    /* address of the remote host */
int servlen;                   /* Unused */
int count;                     /* number of bytes transferred */

/* Build a socket.*/

/* Build the server's address structure. */
servaddr->family = NU_FAMILY_IP6; /* Specifies IPv6 */
servaddr->port = 0;                /* server's port number */

servaddr->id.is_ip_addrs[0] = 0xfe;
servaddr->id.is_ip_addrs[1] = 0x80; /* the server machine's */
servaddr->id.is_ip_addrs[2] = 0;    /* 16-digit IP number */
servaddr->id.is_ip_addrs[3] = 0;
servaddr->id.is_ip_addrs[4] = 0;
servaddr->id.is_ip_addrs[5] = 0;
servaddr->id.is_ip_addrs[6] = 0;
servaddr->id.is_ip_addrs[7] = 0;
servaddr->id.is_ip_addrs[8] = 0;
servaddr->id.is_ip_addrs[9] = 0;
servaddr->id.is_ip_addrs[10] = 0xa;
servaddr->id.is_ip_addrs[11] = 0xb;
servaddr->id.is_ip_addrs[12] = 0xff;
servaddr->id.is_ip_addrs[13] = 0x2;
servaddr->id.is_ip_addrs[14] = 0xc;
servaddr->id.is_ip_addrs[15] = 0xd;

servaddr->name = "server_name"

/* Send the data to the IPv6 node. */
count = NU_Send_To_Raw (socketd, buff, strlen(buff), 0,
                        servaddr, servlen);

/* Check the count. */
if (count < 0)
    printf("bad error occurred.\n\r");

```

## Related Topics

[Net API Functions](#)[NU\\_Recv \(IPv4/IPv6\)](#)[NU\\_Select \(IPv4/IPv6\)](#)[NU\\_Send\\_To \(IPv4/IPv6\)](#)[NU\\_Socket \(IPv4/IPv6\)](#)[NU\\_Getsockopt \(IPv4/IPv6\)](#)[NU\\_Recvmsg \(IPv4/IPv6\)](#)[NU\\_Send \(IPv4/IPv6\)](#)[NU\\_Setsockopt \(IPv4/IPv6\)](#)[NU\\_CMSG\\_DATA \(IPv6\)](#)



## NU\_Set\_Default\_Hop\_Limit (IPv6)

This function configures the default Hop Limit to be put in the IPv6 header of outgoing packets.

### Usage

```
STATUS NU_Set_Default_Hop_Limit (UINT8 default_hop_limit);
```

### Arguments

- default\_hop\_limit

The new IPv6 default Hop Limit for the system. By default, this value is set to the macro `IP6_HOP_LIMIT` in *networking/net6\_cfg.h*.

### Return Values

- NU\_SUCCESS

Upon successful completion. Otherwise, the routine returns an operating system-specific error code.

### Example

```
/* Set the default Hop Limit. */  
if (NU_Set_Default_Hop_Limit(128) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_IPV6\\_UNICAST\\_HOPS \(IPv6\)](#)

## NU\_Set\_Default\_TTL (IPv4)

This function configures the default IP Time To Live to be put in the IP header of outgoing packets.

### Usage

```
STATUS NU_Set_Default_TTL (UINT8 default_ttl);
```

### Arguments

- `default_ttl`  
The new IP default Time To Live for the system. By default, this value is set to the macro `IP_TIME_TO_LIVE` located at *networking/net\_cfg.h*.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_UNAVAILABLE`  
The semaphore could not be obtained to perform the operation.

### Example

```
/* Set the default Time To Live. */  
if (NU_Set_Default_TTL(128) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)

## NU\_Set\_Device\_Hostname

This function changes the host name of the interface associated with the respective interface index.

### Usage

```
STATUS NU_Set_Device_Hostname(CHAR    *name,  
                              UINT32  dev_index);
```

### Arguments

- **name**  
Pointer to the new host name or NU\_NULL to clear the host name.
- **dev\_index**  
The index of the interface being configured.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully.
- **NU\_INVALID\_PARM**  
There is no interface with a matching index.

### Description

The host name was configured initially when the interface was initialized using the metadata option "hostname" and the interface index. The interface's host name is used only when mDNS is enabled to claim ownership of the A and AAAA records for the respective IP address(es) assigned to the interface and to answer A, AAAA and PTR queries for the host name. If the interface has already been configured with an IP address(es) before this routine is invoked, the respective local DNS record(s) will be updated with the new host name, and probing and announcing will be initiated.

If the host name is too long, it will be truncated appropriately in the memory provided by the caller.

### Example

```
STATUS    status;  
  
/* Change the configured hostname. */  
status = NU_Set_Device_Hostname("mentor_host", 1);
```

### Related Topics

[Net API Functions](#)

## NU\_Set\_DHCP\_DUID (IPv4/IPv6)

This function sets the DUID (DHCP Unique Identifier) for the node for use with DHCP.

### Usage

```
STATUS NU_Set_DHCP_DUID (DHCP_DUID_STRUCT *duid_ptr);
```

### Arguments

- **duid\_ptr**  
A pointer to the data structure that contains the DUID information for the node.  
The [DHCP\\_DUID\\_STRUCT](#) data structure is defined in the [Net Data Structures](#) section.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
One of the input parameters is invalid.

### Example

```
DHCP_DUID_STRUCT duid_ptr;  
STATUS status;  
  
/* Set up a DUID-EN type DUID. */  
duid_ptr.duid_type = DHCP_DUID_EN;  
duid_ptr.duid_ent_no = 0x12345678;  
strcpy(duid_ptr.duid_id, "thisstringistheid");  
duid_ptr.duid_id_no_len = strlen(duid_ptr.duid_id);  
  
/* Set the DUID for this node. */  
status = NU_Set_DHCP_DUID(&duid_ptr);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Set_DHCP_DUID.\n");  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>	<a href="#">NU_Dhcp6 (IPv6)</a>
<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>	<a href="#">NU_Get_DHCP_DUID (IPv4/IPv6)</a>
<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>	<a href="#">NU_Set_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## NU\_Set\_DHCP\_IAID (IPv4/IPv6)

This function sets the Identity Association Identifier (IAID) associated with an interface for use with DHCP.

### Usage

```
STATUS NU_Set_DHCP_IAID (UINT32 dev_index,  
                        UINT32 iaid);
```

### Arguments

- `dev_index`  
The index number of the interface for which to set the IAID.
- `iaid`  
A pointer to the new IAID to set on the interface.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_PARM`  
One of the input parameters is invalid.

### Example

```
UINT32          iaid = 0x12345678;  
DHCP6_CLIENT    ds_ptr;  
STATUS          status;  
  
/* Get the index associated with this interface name. */  
ds_ptr.dhcp6_dev_index = NU_IF_NameToIndex("eth0");  
  
/* Get the IA ID associated with this interface. */  
status = NU_Set_DHCP_IAID(ds_ptr.dhcp6_dev_index, iaid);  
  
if (status != NU_SUCCESS)  
{  
    printf("Error at call to NU_Set_DHCP_IAID.\n");  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">DHCP6_Build_Option_Request_Option (IPv6)</a>
<a href="#">DHCP6_Build_User_Class_Option (IPv6)</a>	<a href="#">DHCP6_Build_Vendor_Class_Option (IPv6)</a>
<a href="#">DHCP6_Build_Vendor_Specific_Info_Option (IPv6)</a>	<a href="#">NU_Dhcp6 (IPv6)</a>
<a href="#">NU_Dhcp6_Shutdown (IPv6)</a>	<a href="#">NU_Get_DHCP_IAID (IPv4/IPv6)</a>
<a href="#">DHCP6_Build_IA_NA_Option (IPv6)</a>	

## NU\_Set\_Domain\_Name (IPv4/IPv6)

This function sets the name of the local domain. The domain name is initially specified by the definition of DOMAINNAME found in *networking/net\_cfg.h*. During initialization, DOMAINNAME is copied into the global variable SCK\_Domain\_Name.

### Usage

```
INT NU_Set_Domain_Name (CHAR *name,  
                        INT  name_length);
```

### Arguments

- **name**  
This is a pointer to the new domain name.
- **name\_length**  
Size of the new domain name. The size must be less than NET\_MAX\_DOMAIN\_NAME\_LENGTH.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion the name of the domain will be changed to the value in the name parameter.
- **NU\_INVALID\_PARM**  
There is a problem with one of the parameters. It is likely that name is a null pointer or the name\_length is either 0 or too large for the SCK\_Domain\_Name variable.

### Example

```
CHAR  dname[8] = "New_Name";  
STATUS status;  
  
status = NU_Set_Domain_Name(dname, strlen(dname));  
if (status != NU_SUCCESS)  
{  
    printf("ERROR: failed to set the domain name");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_MX](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Sock\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Peer\\_Name \(IPv4/IPv6\)](#)

[NU\\_Set\\_Host\\_Name \(IPv4/IPv6\)](#)



## NU\_Set\_Host\_Name (IPv4/IPv6)

This function sets the name of the local host. The domain name is initially specified by the definition of HOSTNAME found in *networking/net\_cfg.h*. During initialization, HOSTNAME is copied into the global variable SCK\_Host\_Name.

### Usage

```
INT NU_Set_Host_Name (CHAR *name,  
                     INT  name_length);
```

### Arguments

- **name**  
This is a pointer to the new host name.
- **name\_length**  
Size of the new host name. The size must be less than MAX\_HOST\_NAME\_LENGTH.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion the name of the host will be changed to the value in the name parameter.
- **NU\_INVALID\_PARM**  
There is a problem with one of the parameters. It is likely that name is a null pointer or the name\_length is either 0 or too large for the SCK\_Host\_Name variable.

### Example

```
CHAR    hname[8] = "New_Name";  
STATUS  status;  
  
status = NU_Set_Host_Name(hname, strlen(hname));  
if (status != NU_SUCCESS)  
{  
    printf("ERROR: failed to set the host name");  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_Host\\_MX](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Sock\\_Name \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)

[NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)

[NU\\_Get\\_Peer\\_Name \(IPv4/IPv6\)](#)

[NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)

## NU\_Set\_IP\_Forwarding (IPv4/IPv6)

This function configures the node to start forwarding or stop forwarding packets. Note that the macro `INCLUDE_IP_FORWARDING` in *networking/net\_cfg.h* must be set to `NU_TRUE` to enable packet-forwarding on the node.

### Usage

```
STATUS NU_Set_IP_Forwarding (UINT8 forwarding);
```

### Arguments

- forwarding  
A value of `NU_FALSE` disables forwarding on the node. A value of `NU_TRUE` enables forwarding on the node.

### Return Values

- `NU_SUCCESS`  
Upon successful completion. Otherwise, this routine returns an operating system-specific error code.

### Example

```
/* Configure the node to forward packets. */  
if (NU_Set_IP_Forwarding(NU_TRUE) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_IP\\_Forwarding \(IPv4/IPv6\)](#)

## NU\_Set\_Reasm\_Max\_Size (IPv4)

This function configures the Maximum Reassembly Size for an interface.

### Usage

```
STATUS NU_Set_Reasm_Max_Size (CHAR    *dev_name,  
                             UINT32  reasm_max_size);
```

### Arguments

- **dev\_name**  
A pointer to the name of the interface for which to modify the Maximum Reassembly Size.
- **reasm\_max\_size**  
The new Maximum Reassembly Size for the interface.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
One of the parameters passed in is invalid.

### Description

All fragmented packets received on this interface must be smaller than or equal to this value or they will be silently discarded. When an interface is initialized, the Maximum Reassembly Size is set to the link MTU for the interface.

### Example

```
/* Set the maximum reassembly size. */  
if (NU_Set_Reasm_Max_Size("eth0", 65535) !=  
    NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Get\\_Reasm\\_Max\\_Size \(IPv4\)](#)

## NU\_Shutdown (IPv4/IPv6)

This function allows the application to shut down a specific communication side of a socket. Once a side of the communication has been shut down, it can no longer be accessed by the application.

### Usage

```
STATUS NU_Shutdown (INT socketd,  
                   INT how);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **how**  
Shutdown method.

There are three options for the parameter how:

**SHUT\_RD**

Closes the read half of the communication.

**SHUT\_WR**

Closes the write half of the communication.

**SHUT\_RDWR**

Closes both the read and write halves of the communication.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket descriptor passed in does not reflect a possible socket in the system.
- **NU\_UNAVAILABLE**  
The semaphore could not be obtained to perform the operation.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_PORT**  
The port associated with the socket is invalid.

### Example

```
INT      socketd;  
STATUS   status;  
.
```

```
.  
.  
/* At this point a socket connection to a peer has been established. */  
status = NU_Shutdown(socketd, SHUT_RD);  
  
/* The value of status can be checked and the appropriate action taken. */
```

## Related Topics

[Net API Functions](#)

[NU\\_Clear\\_Internal\\_Log](#)

[NU\\_Abort \(IPv4/IPv6\)](#)

## NU\_Socket (IPv4/IPv6)

This function is responsible for establishing a new socket descriptor and defining the type of communication protocol to be established. It must be called by both the client and the server whether a connection-oriented or connectionless transfer is being established.

### Usage

```
STATUS NU_Socket (INT16 family,  
                  INT16 type,  
                  INT16 protocol);
```

### Arguments

- **family**  
The type of communication that can be invoked on the socket – NU\_FAMILY\_IP for an IPv4-only socket or NU\_FAMILY\_IP6 for an IPv6/IPv4 socket.
- **type**  
Valid entries for the type parameter include: NU\_TYPE\_STREAM, which specifies TCP Protocol, NU\_TYPE\_DGRAM, which specifies UDP Protocol, and NU\_TYPE\_RAW, which specifies IP Protocol.
- **protocol**  
The protocol parameter should be set to NU\_NONE for NU\_TYPE\_STREAM and NU\_TYPE\_DGRAM sockets. For NU\_TYPE\_RAW sockets, valid values are IPPROTO\_HELLO, IPPROTO\_OSPF, IPPROTO\_RAW, and 0. Refer to the following tables for descriptions of these values.

**Table 2-38. NU\_TYPE\_STREAM or NU\_TYPE\_DGRAM Type**

NU_TYPE_STREAM or NU_TYPE_DGRAM type	Protocol
NU_NONE	The protocol field is not used.

**Table 2-39. NU\_TYPE\_RAW Type**

NU_TYPE_RAW type	Protocols
IPPROTO_HELLO	HELLO routing protocol.
IPPROTO_OSPF	OSPF routing protocol.
IPPROTO_RAW	Raw IP protocol.
0	Raw IP wildcard protocol.

## Return Values

- A socket descriptor with a value greater than or equal to 0. It is used in other routines' parameter lists under the label socketd.

Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:

- **NU\_NO SOCK MEMORY**  
The memory for the sockets is depleted.
- **NU\_INVALID\_PROTOCOL**  
The specified protocol is invalid.

## Related Topics

[Net API Functions](#)

[NU\\_Clear\\_Internal\\_Log](#)

[NU\\_Getsockopt \(IPv4/IPv6\)](#)

[NU\\_Recv \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_Send \(IPv4/IPv6\)](#)

[NU\\_Send\\_To \(IPv4/IPv6\)](#)

[NU\\_Setsockopt \(IPv4/IPv6\)](#)

[NU\\_Accept \(IPv4/IPv6\)](#)

[NU\\_Bind \(IPv4/IPv6\)](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

[NU\\_Listen \(IPv4/IPv6\)](#)

[NU\\_Recvmsg \(IPv4/IPv6\)](#)

[NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Sendmsg \(IPv4/IPv6\)](#)

[NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)

[NU\\_Shutdown \(IPv4/IPv6\)](#)

## NU\_Setsockopt (IPv4/IPv6)

This function sets and clears socket options for a specified socket.

### Usage

```
STATUS NU_Setsockopt (INT  socketd,  
                     INT  level,  
                     INT  optname,  
                     VOID *optval,  
                     INT  optlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **level**  
Specifies the protocol level. Valid entries for this parameter are; SOL\_SOCKET, IPPROTO\_IP, IPPROTO\_IPV6, IPPROTO\_TCP and IPPROTO\_UDP.
- **optname**  
Specifies an option. Valid options are:

SO_BROADCAST	SO_LINGER	IPV6_RECVHOPOPTS
SO_RCVBUF		IPV6_RECVPKTINFO
IP_ADD_MEMBERSHIP		IPV6_RECVRTHDR
IP_DROP_MEMBERSHIP		IPV6_RECVTCLASS
IP_MULTICAST_TTL		IPV6_RTHDR
IP_MULTICAST_IF		IPV6_RTHDRDSTOPTS
IP_BROADCAST_IF		IPV6_TCLASS
IP_RECVIFADDR		IPV6_UNICAST_HOPS
IP_HDRINCL		IPV6_V6ONLY
IP_TOS		TCP_FIRST_PROBE_TIMEOUT
IP_TTL		TCP_PROBE_TIMEOUT
TCP_NODELAY		TCP_MAX_PROBES
SO_KEEPALIVE		TCP_MSL
SO_REUSEADDR		TCP_FIRST_RTO
IPV6_CHECKSUM		TCP_MAX_RTO
IPV6_DSTOPTS		TCP_MAX_R2
IPV6_HOPOPTS		TCP_MAX_SYN_R2
IPV6_JOIN_GROUP		TCP_DELAY_ACK
IPV6_LEAVE_GROUP		TCP_KEEPALIVE_WAIT
IPV6_MULTICAST_HOPS		TCP_KEEPALIVE_R2
IPV6_MULTICAST_IF		TCP_CONGESTION_CTRL
IPV6_NEXTHOP		TCP_CFG_SACK
IPV6_PKTINFO		TCP_WINDOWSCALE
IPV6_RECVDSTOPTS		TCP_RCV_WINDOWSIZE
IPV6_RECVHOPLIMIT		TCP_TIMESTAMP



Table 2-40, Table 2-41, Table 2-42, Table 2-43, and Table 2-44 list the levels and the associated options that are currently supported by NU\_Setsockopt.

**Table 2-40. SOL\_SOCKET Level Options**

SOL_SOCKET Level	Family	Options
SO_BROADCAST	IPv4	Set the broadcast status of a socket. When a socket is created, the ability to send broadcasts is enabled by default.
SO_LINGER	IPv4/IPv6	Set the linger option. This option is disabled by default.
SO_REUSEADDR	IPv4/IPv6	Set the option to reuse a port for a unique IP address.
SO_RCVBUF	IPv4/IPv6	<p>Set the TCP Window Size for the local side of the connection on a socket. This value must be greater than zero, and the socket option must be set before a call to NU_Connect or NU_Listen for the socket.</p> <p>This socket option has been deprecated in favor of TCP_RCV_WINDOWSIZE in order to take advantage of the larger window size made possible by the use of the TCP Window Scale option.</p>

**Table 2-41. IPPROTO\_IP Level Options**

IPPROTO_IP Level	Family	Options
IP_ADD_MEMBERSHIP	IPv4	Join a multicast group. Note that setting this socket option has been deprecated in favor of the routine NU_IP_Multicast_Listen().
IP_DROP_MEMBERSHIP	IPv4	Leave a multicast group. Note that setting this socket option has been deprecated in favor of the routine NU_IP_Multicast_Listen().
IP_HDRINCL	IPv4	This option, when set, specifies that the application layer is responsible for providing the IPv4 header. When clear, the application will only pass data to the stack, which will then append an IPv4 header. This option is only valid when using the RAW IP services.

**Table 2-41. IPPROTO\_IP Level Options (cont.)**

IPPROTO_IP Level	Family	Options
IP_MULTICAST_TTL	IPv4	Change the default TTL (Time To Live) for multicast packets sent on this socket. The default multicast TTL is specified by the macro <code>IP_DEFAULT_MULTICAST_TTL</code> in the file <i>networking/net_cfg.h</i> . A default value of 1 prevents routers from forwarding multicast datagrams beyond the local network.
IP_MULTICAST_IF	IPv4	Change the interface to be used for sending multicast datagrams. The IP address of the interface is set in the <code>optval</code> parameter as a 4-byte array. By default, the stack searches the Routing Table for the interface to use.
IP_BROADCAST_IF	IPv4	Change the interface to be used for sending broadcast datagrams. The IP address of the interface is set in the <code>optval</code> parameter as a 4-byte array. By default, the first non-loopback interface is used.
IP_RECVIFADDR	IPv4	Enable the receive interface address option. This option enables the storing of the receive interface upon calls to <code>NU_Recv_From()</code> . After a call to <code>NU_Recv_From()</code> a call to <code>NU_Recv_IF_Addr()</code> can be made in order to obtain the IP address of the interface that received the datagram. This is useful when receiving multicast and broadcast packets that do not have a specific destination IP address. Zero disables and non-zero enables this option.
IP_TTL	IPv4	Set the Time To Live to be used in the header of outgoing IP packets using this socket. The default value of this parameter is set according to the default TTL for the system, which is specified by the macro <code>IP_TIME_TO_LIVE</code> by default, but can be modified using the routine <code>NU_Set_Default_TTL()</code> .
IP_TOS	IPv4	Retrieve the Type of Service/Differentiated Services Code Point for the specified socket.

**Table 2-42. IPPROTO\_IPv6 Level Options**

IPPROTO_IPv6 Level	Family	Options
IPV6_CHECKSUM	IPv6	Sets the socket option for enabling or disabling the socket option for the IP layer to compute and store the checksum for outgoing RAW IPv6 packets and verify the checksum for incoming RAW IPv6 packets.
IPV6_DSTOPTS	IPv6	Sets the Destination Options extension header Sticky Option for this socket. The Destination Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Destination Options extension header as ancillary data.
IPV6_HOPOPTS	IPv6	Sets the Hop-By-Hop Options extension header Sticky Option for this socket. The Hop-By-Hop Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Hop-By-Hop Options extension header as ancillary data.
IPV6_JOIN_GROUP	IPv6	Joins a multicast group on the specified interface.
IPV6_LEAVE_GROUP	IPv6	Leaves a multicast group on the specified interface.
IPV6_MULTICAST_HOPS	IPv6	Sets the hop limit to use in multicast packets transmitted using this socket.
IPV6_MULTICAST_IF	IPv6	Sets the interface to use for multicast packets transmitted using this socket.
IPV6_NEXTHOP	IPv6	Sets the Sticky Option for the Next-Hop to be used with all packets sent from this socket. Note that this Sticky Option can be overridden on a per-packet basis with a call to NU_Sendmsg that specifies a different Next-Hop as ancillary data.
IPV6_PKTINFO	IPv6	Sets the source address and/or outgoing interface index Sticky Option on the socket.
IPV6_RECVDSTOPTS	IPv6	Enables or disables the value of the socket option to receive the Destination Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

**Table 2-42. IPPROTO\_IPv6 Level Options (cont.)**

IPPROTO_IPv6 Level	Family	Options
IPV6_RECVHOPLIMIT	IPv6	Sets the socket option to return the hop limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVHOPOPTS	IPv6	Enables or disables the value of the socket option to receive the Hop-By-Hop Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVPKTINFO	IPv6	Retrieves the value of the socket option to return the receive interface and destination address of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVRTHDR	IPv6	Enables or disables the value of the socket option to receive the Routing Header of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RECVTCLASS	IPv6	Sets the socket option to return the Traffic Class of the most recently received packet on this socket. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.
IPV6_RTHDR	IPv6	Sets the Routing Header Sticky Option for this socket. The Routing Header set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Routing Header as ancillary data.
IPV6_RTHDRDSTOPTS	IPv6	Sets the Destination Options extension header Sticky Option for the Destination Options to precede the Routing Header for this socket. The Destination Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Destination Options extension header as ancillary data.

**Table 2-42. IPPROTO\_IPv6 Level Options (cont.)**

IPPROTO_IPv6 Level	Family	Options
IPV6_TCLASS	IPv6	Sets the Traffic Class Sticky Option on the socket. The Traffic Class set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU_Sendmsg specifying a different Traffic Class as ancillary data.
IPV6_UNICAST_HOPS	IPv6	Controls the hop limit used in outgoing unicast IPv6 packets.
IPV6_V6ONLY	IPv6	Sets the socket option for a socket of family type NU_FAMILY_IP6 to be used for IPv6 communications only. By setting this socket option, the socket can be used to send and receive IPv6 packets only.

**Table 2-43. IPPROTO\_TCP Level Options**

IPPROTO_TCP Level	Family	Option
SO_KEEPAIVE	IPv4/IPv6	Enable TCP Keep-Alives on this socket.
TCP_CFG_DSACK	IPv4/IPv6	Toggle support for the Duplicate Selective Acknowledgement (D-SACK) support on a TCP socket. A value of zero disables D-SACKs for the socket. A non-zero value enables D-SACKs for the socket. D-SACK support is enabled by default for all sockets when D-SACK support is compiled into the system via NET_INCLUDE_DSACK in <i>net_cfg.h</i> . This value can be set at any point during the lifetime of a socket. The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.

**Table 2-43. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_CFG_SACK	IPv4/IPv6	Toggle support for the Selective Acknowledgement (SACK) support on a TCP socket. A value of zero disables SACKs for the socket. A non-zero value enables SACKs for the socket. SACK support is enabled by default for all sockets when SACK support is compiled into the system via <code>NET_INCLUDE_SACK</code> in <i>net_cfg.h</i> . This value can only be set before a connection has been established over the socket. The configured value is inherited by all child sockets created from the parent socket at the call to <code>NU_Accept</code> during run-time.
TCP_CONGESTION_CTRL	IPv4/IPv6	Toggle support for the TCP Congestion Control Algorithm on a TCP socket. A value of zero disables Congestion Control for the socket. A non-zero value enables Congestion Control for the socket. Congestion Control is enabled by default for all sockets when Congestion Control support is compiled into the system via <code>INCLUDE_CONGESTION_CONTROL</code> in <i>net_cfg.h</i> . This value can only be set before a connection has been established over the socket. The configured value is inherited by all child sockets created from the parent socket at the call to <code>NU_Accept</code> during run-time.
TCP_DELAY_ACK	IPv4/IPv6	Set the amount of time to delay TCP ACKs on a TCP socket at run-time. This timeout value is set to <code>TCP_ACK_TIMEOUT</code> by default when a new TCP socket is created. <code>TCP_ACK_TIMEOUT</code> can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to <code>NU_Accept</code> during run-time.

**Table 2-43. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_FIRST_PROBE_TIMEOUT	IPv4/IPv6	Set the timeout value for sending the initial Zero Window probe on a TCP socket at run-time. This timeout value is set to PROBE_TIMEOUT by default when a new TCP socket is created. PROBE_TIMEOUT can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_FIRST_RTO	IPv4/IPv6	Set the value of the first retransmission timeout on a TCP socket at run-time. This timeout value is set to TCP_RTTFDFT by default when a new TCP socket is created. TCP_RTTFDFT can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_KEEPALIVE_R2	IPv4/IPv6	Set the maximum number of unanswered keep-alive probes to transmit on a TCP socket at run-time. This value is set to TCP_MAX_KEEPALIVES by default when a new TCP socket is created. TCP_MAX_KEEPALIVES can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_KEEPALIVE_WAIT	IPv4/IPv6	Set the amount of time to remain idle on a TCP socket before invoking the TCP keep-alive protocol at run-time. This value is set to TCP_KEEPALIVE_INTERVAL by default when a new TCP socket is created. TCP_KEEPALIVE_INTERVAL can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.

**Table 2-43. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_MAX_R2	IPv4/IPv6	Set the number of times to retransmit a data segment before closing the connection on a TCP socket at run-time. This value is set to MAX_RETRANSMITS by default when a new TCP socket is created. MAX_RETRANSMITS can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_MAX_RTO	IPv4/IPv6	Set the value of the maximum retransmission timeout on a TCP socket at run-time. This timeout value is set to MAXRTO by default when a new TCP socket is created. MAXRTO can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_MAX_SYN_R2	IPv4/IPv6	Set the number of times to retransmit a SYN packet before closing the connection on a TCP socket at run-time. This timeout value is set to MAX_RETRANSMITS by default when a new TCP socket is created. MAX_RETRANSMITS can be configured at compile-time in the file <i>net_cfg.h</i> .
TCP_PROBE_TIMEOUT	IPv4/IPv6	Set the maximum timeout value for sending successive Zero Window probes on a TCP socket at run-time. This timeout value is set to MAX_PROBETIMEOUT by default when a new TCP socket is created. MAX_PROBETIMEOUT can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.



**Table 2-43. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_MAX_PROBES	IPv4/IPv6	Set the maximum number of Zero Window probes to be transmitted before giving up on window probing on a TCP socket at run-time. This value is set to TCP_MAX_PROBE_COUNT by default when a new TCP socket is created. TCP_MAX_PROBE_COUNT can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_MSL	IPv4/IPv6	Set the amount of time to wait before reusing a port / IP address combination on a TCP socket at run-time. This timeout value is set to WAITTIME by default when a new TCP socket is created. WAITTIME can be configured at compile-time in the file <i>net_cfg.h</i> . The configured value is inherited by all child sockets created from the parent socket at the call to NU_Accept during run-time.
TCP_NODELAY	IPv4/IPv6	TCP sockets, by default, utilize the Nagle Algorithm. The Nagle Algorithm attempts to merge contiguous small TCP packets into a single large TCP packet. It does this by delaying the transmission of small packets. This better utilizes the network bandwidth. However, it is often desirable for data to be transmitted immediately. The TCP_NODELAY option can be used to control the disposition of the Nagle Algorithm. A value of 1 will eliminate the delay, for example, turn the Nagle Algorithm off. A value of 0 will enable the delay, for example, turn the Nagle Algorithm on.

**Table 2-43. IPPROTO\_TCP Level Options (cont.)**

IPPROTO_TCP Level	Family	Option
TCP_RCV_WINDOWSIZE	IPv4/IPv6	Configure the value of the local receive window for the socket. This value is set to WINDOW_SIZE by default for all new sockets. The value must be greater than zero and less than TCP_MAX_WINDOW_SIZE. If a value greater than 16-bits is configured, you must also configure the shift count for the socket using the TCP_WINDOWSCALE socket option to ensure the value will fit in the 16-bit field of the TCP header. The value must be configured before a connection has been made on the socket. The configured value is inherited by all child sockets created from the parent socket at run-time.
TCP_TIMESTAMP	IPv4/IPv6	Toggle support for the TCP Timestamp option on a TCP socket. A value of zero disables the Timestamp option for the socket. A non-zero value enables the Timestamp option. Support for the TCP Timestamp option is enabled by default for all sockets when the TCP Timestamp option support is compiled into the system via NET_INCLUDE_TIMESTAMP in <i>net_cfg.h</i> . The value must be configured before a connection has been made on the socket. The configured value is inherited by all child sockets created from the parent socket at run-time.
TCP_WINDOWSCALE	IPv4/IPv6	Toggle support for the TCP Window Scale option on a TCP socket. A value of -1 disables the Window Scale option for the socket. A non-zero value enables the Window Scale option, and the shift count for the option is set to the respective value. Support for the TCP Window Scale option is enabled by default for all sockets when TCP Window Scale option support is compiled into the system via NET_INCLUDE_WINDOWSCALE in <i>net_cfg.h</i> . The value must be configured before a connection has been made on the socket. The configured value is inherited by all child sockets created from the parent socket at run-time.

**Table 2-44. IPPROTO\_UDP Level Options**

IPPROTO_UDP Level	Family	Options
UDP_NOCHECKSUM	IPv4	This function enables or disables the computation of the UDP checksum for outgoing IPv4 packets and the verification of the UDP checksum for incoming IPv4 packets transmitted and received using this socket. A value of zero disables the UDP_NOCHECKSUM option on the socket; enabling the computation and verification of the UDP checksum for the socket. A non-zero value enables the UDP_NOCHECKSUM option on the socket; disabling the computation and verification of the UDP checksum for the socket.

- **optval**  
Pointer to the new value for the option.
- **optlen**  
Specifies the size in bytes of the location pointed to by optval.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion, the option specified by optname will be set to the value in optval.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_LEVEL**  
The value specified in the level parameter is invalid.
- **NU\_INVALID**  
Either the optval or optlen parameter has problems.
- **NU\_MEM\_ALLOC**  
If there is not enough memory to complete the request.
- **NU\_ADDRINUSE**  
Membership already exists in the specified multicast group.

- **NU\_INVALID\_OPTION**

The value specified in the optname parameter is invalid.

- **NU\_UNAVAILABLE**

The option must be enabled in the system before being set on a socket.

For an example demonstrating the use of the NU\_Setsockopt service, see the [NU\\_Getsockopt \(IPv4/IPv6\)](#) service call.

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_RCVBUF (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_ADD\_MEMBERSHIP (IPv4)

This function joins a multicast group for a socket.

### Usage

```
INT32 NU_Setsockopt_IP_ADD_MEMBERSHIP (INT      socketd,  
                                       IP_MREQ *mreq_in);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **mreq\_in**  
Pointer to the multicast data structure, where the member `sck_multiaddr` contains the multicast address group to join, and `sck_inaddr` contains the address of the device on which to join the multicast group.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- **NU\_MEM\_ALLOC**  
Insufficient memory to join the group.
- **NU\_INVALID**  
One of the parameters provided is invalid.
- **NU\_ADDRINUSE**  
The multicast address is already registered for the interface.

### Description

It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_IP` and type of `IP_ADD_MEMBERSHIP`. Note that use of this function has been deprecated in favor of the routine `NU_IP_Multicast_Listen()`.

### Example

```
INT      socketd;                               /* socket descriptor */  
IP_MREQ  mreq_in;  
UINT8    multi_addr[4] = {224, 0, 0, 1};  
UINT8    dev_addr[4]   = {192, 168, 1, 21};
```

```

/* Build a socket.*/

/* Copy the multicast IP address to join into the data
 * structure.
 */
memcpy(&mreq_in.sck_multiaddr, multi_addr, 4);

/* Copy the IP address of the device on which to join the
 * multicast address into the data structure.
 */
memcpy(&mreq_in.sck_inaddr, dev_addr, 4);

/* Join the specified multicast address on the specified device.
 */
if (NU_Setsockopt_IP_ADD_MEMBERSHIP(socketd, &mreq_in) != NU_SUCCESS)
{
    /* Fatal error, abort here. */
}

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPAIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>
<a href="#">NU_IP_Multicast_Listen (IPv4)</a>	

## NU\_Setsockopt\_IP\_DROP\_MEMBERSHIP (IPv4)

This function leaves a multicast group for a socket.

### Usage

```
INT32 NU_Setsockopt_IP_DROP_MEMBERSHIP (INT      socketd,  
                                         IP_MREQ *mreq_in);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **mreq\_in**  
Pointer to the multicast data structure, where the member `sck_multiaddr` contains the multicast address group to leave, and `sck_inaddr` contains the address of the device on which to leave the multicast group.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- **NU\_MEM\_ALLOC**  
Insufficient memory.
- **NU\_INVALID**  
One of the parameters provided is invalid.

### Description

It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_IP` and type of `IP_DROP_MEMBERSHIP`. Note that use of this function has been deprecated in favor of the routine `NU_IP_Multicast_Listen()`.

### Example

```
INT      socketd;                /* socket descriptor */  
IP_MREQ  mreq_in;  
UINT8    multi_addr[4] = {224, 0, 0, 1};  
UINT8    dev_addr[4] = {192, 168, 1, 21};  
  
/* Build a socket.*/  
  
/* Copy the multicast IP address to leave in the data structure.
```



```

    */
    memcpy(&mreq_in.sck_multiaddr, multi_addr, 4);

    /* Copy the IP address of the device on which to leave the
     * multicast address into the data structure.
     */
    memcpy(&mreq_in.sck_inaddr, dev_addr, 4);
    /* Leave the specified multicast address on the specified device.
     */
    if (NU_Setsockopt_IP_DROP_MEMBERSHIP(socketd, &mreq_in) != NU_SUCCESS)
    {
        /* Fatal error, abort here. */
    }

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>
<a href="#">NU_IP_Multicast_Listen (IPv4)</a>	

## NU\_Setsockopt\_IP\_BROADCAST\_IF (IPv4)

This function sets the broadcast interface to use for outgoing packets on a socket.

### Usage

```
STATUS NU_Setsockopt_IP_BROADCAST_IF (INT    socketd,  
                                       UINT8  *dev_address);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **dev\_address**  
A pointer to the IPv4 address associated with the broadcast interface to use for outgoing packets.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- **NU\_INVAL**  
The address passed in does not reference an interface on the node.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.

### Description

It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_IP` and type of `IP_BROADCAST_IF`.

### Example

```
INT    socketd; /* socket descriptor */  
UINT8  dev_address[4] = {192, 168, 1, 21};  
  
/* Build a socket.*/  
  
/* Set the broadcast interface for the socket */  
if (NU_Setsockopt_IP_BROADCAST_IF(socketd, dev_address) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_HDRINC (IPv4)

This function sets the flag to indicate that the IP layer should build the IPv4 header for RAW IP packets. It performs the same functionality as a call to the function NU\_Setsockopt with a level of IPPROTO\_IP and type of IP\_HDRINCL.

### Usage

```
STATUS NU_Setsockopt_IP_HDRINCL (INT  socketd,  
                                INT16 opt_val);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **opt\_val**  
A value of zero indicates that the IP layer should build an IPv4 header for RAW IP packets transmitted using this socket. A non-zero value indicates that the IP layer should not build an IPv4 header for RAW IP packets transmitted using this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Example

```
INT socketd; /* socket descriptor */  
  
/* Build an IPRAW socket.*/  
  
/* Set the flag to indicate that the application layer will  
 * build the IP header for IPRAW packets sent on this socket.  
 */  
if (NU_Setsockopt_IP_HDRINCL(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_MULTICAST\_IF (IPv4)

This function sets the interface to use when transmitting multicast packets when sending multicast packets on the specified socket. It performs the same functionality as a call to the function NU\_Setsockopt with a level of IPPROTO\_IP and type of IP\_MULTICAST\_IF.

### Usage

```
STATUS NU_Setsockopt_IP_MULTICAST_IF (INT    socketd,  
                                     UINT8  *dev_address);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **dev\_address**  
A pointer to the IPv4 address of the interface to use when transmitting multicast packets using this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.
- **NU\_INVALID**  
There is no port structure associated with this socket.
- **NU\_MEM\_ALLOC**  
Insufficient memory.

### Example

```
INT    socketd; /* socket descriptor */  
UINT8  dev_address[4] = {192, 168, 1, 21};  
  
/* Build a socket.*/  
  
/* Set the interface to use when transmitting multicast packets  
 * using this socket.  
 */  
if (NU_Setsockopt_IP_MULTICAST_IF(socketd, dev_address) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_MULTICAST\_TTL (IPv4)

This function sets the IPv4 Time To Live to put in the IP header of outgoing multicast packets transmitted using this socket. It performs the same functionality as a call to the function NU\_Setsockopt with a level of IPPROTO\_IP and type of IP\_MULTICAST\_TTL.

### Usage

```
STATUS NU_Setsockopt_IP_MULTICAST_TTL (INT    socketd,  
                                       UINT8  sck_ttl);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **sck\_ttl**  
The Time To Live to put in the IPv4 header of multicast packets is sent using this socket. By default, this value is set to the macro IP\_DEFAULT\_MULTICAST\_TTL located at *networking/net\_cfg.h*.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.
- **NU\_MEM\_ALLOC**  
Insufficient memory.

### Example

```
INT    socketd; /* socket descriptor */  
UINT8  ttl = 20;  
  
/* Build a socket.*/  
  
/* Set the TTL to put in the IP header of outgoing multicast  
 * packets sent using this socket.  
 */  
if (NU_Setsockopt_IP_MULTICAST_TTL(socketd, ttl) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_RECVIFADDR (IPv4)

This function enables and disables the receive interface option, which allows the Application Layer to determine which interface received the most recently received packet. It performs the same functionality as a call to the function NU\_Setsockopt with a level of IPPROTO\_IP and type of IP\_RECVIFADDR.

### Usage

```
STATUS NU_Setsockopt_IP_RECVIFADDR (INT      socketd,  
                                     UINT16  opt_val);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **opt\_val**  
A value of zero disables the receive interface option on the socket. A non-zero value enables the receive interface option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_NOT\_CONNECTED**  
A connection does not exist on the specified socket.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value or it had not been previously allocated via the NU\_Socket call.

### Example

```
INT  socketd; /* socket descriptor */  
  
/* Build a socket.*/  
  
/* Enable the receive interface option on the socket. */  
if (NU_Setsockopt_IP_RECVIFADDR(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_TOS (IPv4)

This function sets the default Type of Service/Differentiated Services Code Point to be put in the IPv4 header of outgoing packets using this socket based on the specifications set forth in RFC 2474. This routine performs the same functionality as a call to the function NU\_Setsockopt with a level of IPPROTO\_IP and type of IP\_TOS.

### Usage

```
STATUS NU_Setsockopt_IP_TOS (INT    socketd,  
                             UINT8  sck_tos);
```

### Arguments

- **socketd**  
Specifies an IPv4 socket descriptor.
- **sck\_tos**  
The Type of Service/Differentiated Services Code Point value to be put into outgoing packets using this socket. Since RFC 2474 uses only the first six bits of the TOS byte, this value must be between 0 and 63, inclusive.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.
- **NU\_INVALID**  
The value of sck\_tos is invalid.

### Example

```
INT    socketd; /* socket descriptor */  
UINT16 sck_tos = 20;  
  
/* Build a socket.*/  
  
/* Set the TOS/DSCP of outgoing packets using this socket. */  
if (NU_Setsockopt_IP_TOS(socketd, sck_tos) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IP\_TTL (IPv4)

This function sets the default Time To Live to be put in the IP header of outgoing packets using this socket. It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_IP` and type of `IP_TTL`.

### Usage

```
STATUS NU_Setsockopt_IP_TTL (INT      socketd,  
                             UINT16  sck_ttl);
```

### Arguments

- `socketd`

Specifies an IPv4 socket descriptor.

- `sck_ttl`

The Time To Live to put in outgoing packets using this socket. By default, the TTL for a socket is set to the default TTL for the system, as specified by the macro `IP_TIME_TO_LIVE` and changed dynamically by the Application Layer using the routine `NU_Set_Default_TTL()`.

### Return Values

- `NU_SUCCESS`

Upon successful completion.

- `NU_MEM_ALLOC`

Insufficient memory.

### Example

```
INT      socketd; /* socket descriptor */  
UINT16  sck_ttl = 20;  
  
/* Build a socket.*/  
  
/* Set the Time To Live of outgoing packets using this socket. */  
if (NU_Setsockopt_IP_TTL(socketd, sck_ttl) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>
<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IPV6\_CHECKSUM (IPv6)

This function sets the socket option for enabling or disabling the socket option for the IP layer to compute and store the checksum for outgoing RAW IPv6 packets and verify the checksum for incoming RAW IPv6 packets.

### Usage

```
STATUS NU_Setsockopt_IPV6_CHECKSUM (INT socketd,  
                                   INT optval);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
If enabling the socket option, optval is the positive even integer offset into the user data where the checksum is located. If disabling the socket option, optval is -1. This socket option is disabled by default.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid. There is either no socket structure associated with the socket descriptor, the socket is not an IPv6 RAW socket, or the socket is an ICMPv6 socket.
- **NU\_INVALID**  
optval is an odd value.

### Description

This routine is helpful for enabling the IP layer to compute the checksum of proprietary protocol types.

An attempt to set this socket option for a RAW ICMPv6 socket or a socket other than an IPv6 RAW socket will fail.

### Example

```
INT socket;  
  
/* Create an IPv6 RAW socket. */  
. . .  
  
/* Set the socket option to compute the checksum which is  
 * located two bytes into the user's data buffer.  
 */  
NU_Setsockopt_IPV6_CHECKSUM(socket, 2);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_DSTOPTS (IPv6)

This function sets the Destination Options extension header Sticky Option for this socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_DSTOPTS (INT          socketd,  
                                   struct ip6_dest *dst_hdr,  
                                   INT          length);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **dst\_hdr**  
A pointer to the IPv6 Destination Options extension header built by the application. Note that this data structure is built by the application using the following API routines: [NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#), and [NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#).
- **length**  
The length of the ip6\_dest structure. A value of zero disables the Destination Options Sticky Option on this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket, or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID**  
The value of the ip6\_dest structure is NU\_NULL, but the value of the length parameter is not zero.

### Description

The Destination Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to [NU\\_Sendmsg](#) specifying a different Destination Options extension header as ancillary data.

### Example

```
INT      socket;  
INT      currentlen;  
UINT32   extlen;
```

```

VOID      *databuf;
INT       offset;
UINT8     value1;
UINT16    value2;
UINT32    value4;
UINT32    value8;
struct ip6_dest *dst_hdr;
/** Build Two Destination Options **/

/* Determine the amount of memory for the Destination Options. */
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X, 12,
                                8, NU_NULL);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y, 7,
                                4, NU_NULL);
currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID**)&dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure. */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for 4-octet field */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof(value4));

/* Insert value 0x0102030405060708 for 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof(value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);
/* Insert value 0x01 for 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof(value1));

/* Insert value 0x1331 for 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof(value2));

/* Insert value 0x01020304 for 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof(value4));
currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);

/* Set the Destination Options Sticky Option to use with all
 * packets sent on this socket.

```

```

    */
    NU_Setsockopt_IPV6_DSTOPTS(socket, dst_hdr, currentlen);

```

**Related Topics**[Net API Functions](#)[NU\\_Setsockopt\\_IPV6\\_HOPOPTS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_LEAVE\\_GROUP \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_MULTICAST\\_IF \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_PKTINFO \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVHOPLIMIT \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVPKTINFO \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVTCLASS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RTHDRDSTOPTS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_UNICAST\\_HOPS \(IPv6\)](#)[NU\\_Setsockopt \(IPv4/IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_CHECKSUM \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_JOIN\\_GROUP \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_MULTICAST\\_HOPS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_NEXTHOP \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVDSTOPTS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVHOPOPTS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RECVRTHDR \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_RTHDR \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_TCLASS \(IPv6\)](#)[NU\\_Setsockopt\\_IPV6\\_V6ONLY \(IPv6\)](#)

## NU\_Setsockopt\_IPV6\_HOPOPTS (IPv6)

This function sets the Hop-By-Hop Options extension header Sticky Option for this socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_HOPOPTS (INT          socketd,  
                                   struct ip6_hbh *hbh_hdr,  
                                   INT          length);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **hbh\_hdr**  
A pointer to the IPv6 Hop-By-Hop Options extension header built by the application. The data structure [ip6\\_hbh](#) is described in the [Net Data Structures](#) section. Note that this data structure is built by the application using the following API routines: [NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#), and [NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#).
- **length**  
The length of the [ip6\\_hbh](#) structure. A value of zero disables the Hop-By-Hop Options Sticky Option on this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket, or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVAL**  
The [ip6\\_hbh](#) structure is [NU\\_NULL](#), but the value of the length parameter is not zero.

### Description

The Hop-By-Hop Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to [NU\\_Sendmsg](#) specifying a different Hop-By-Hop Options extension header as ancillary data.

### Example

```
INT      socket;  
INT      currentlen;  
VOID     *databuf;
```

```
INT      offset;
UINT8    value1;
UINT16   value2;
UINT32   value4;
UINT32   value8;
struct ip6_hbh*hbh_hdr;

/** Build Two Hop-By-Hop Options. */

/* Determine the amount of memory for the Hop-By-Hop Options. */
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X, 12,
                                8, NU_NULL);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y, 7,
                                4, NU_NULL);
currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Hop-By-Hop Options. */
NU_Allocate_Memory(&System_Memory, (VOID**)&hbh_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Hop-By-Hop Options data structure. */
currentlen = NU_Inet6_Opt_Init(hbh_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(hbh_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof(value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof(value8));

currentlen = NU_Inet6_Opt_Append(hbh_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for an 1-octet field */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof(value1));

/* Insert value 0x1331 for a 2-octet field */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof(value2));

/* Insert value 0x01020304 for a 4-octet field */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof(value4));

currentlen = NU_Inet6_Opt_Finish(hbh_hdr, extlen, currentlen);
```

```

/* Set the Hop-By-Hop Options Sticky Option to use with all
 * packets sent on this socket.
 */
NU_Setsockopt_IPV6_HOPOPTS(socket, hbh_hdr, currentlen);

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_JOIN\_GROUP (IPv6)

This function joins an IPv6 multicast group on a socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_JOIN_GROUP (INT socketd,  
                                       struct IP6_MREQ *mreq);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **mreq**  
A pointer to the IPv6 multicast structure containing the multicast address to join and the interface index of the interface on which to join the multicast group.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the multicast structure for the socket.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVAL**  
The multicast address is invalid, the interface is not enabled for multicast, or all multicast positions for the socket are filled.
- **NU\_INVALID\_PARM**  
mreq is invalid.
- **NU\_ADDRINUSE**  
The interface is already a member of the multicast group.

### Example

```
INT socket;  
struct IP6_MREQ mreq;  
  
/* Create a socket. */  
...  
  
/* Get the interface index by name. */  
mreq.ip6_mreq_dev_index = NU_IF_NameToIndex(if_name);  
  
/* Copy the multicast group into the structure. */  
NU_BLOCK_COPY(&mreq.ip6_mreq_multiaddr, m_addr, IP6_ADDR_LEN);
```



```
/* Join the multicast group. */
NU_Setsockopt_IPV6_JOIN_GROUP(socket, &mreq);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_LEAVE\_GROUP (IPv6)

This function leaves an IPv6 multicast group on a socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_LEAVE_GROUP (INT socketd,  
                                       struct IP6_MREQ *mreq);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **mreq**  
A pointer to the IPv6 multicast structure containing the multicast address to leave and the interface index of the interface on which to leave the multicast group.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVAL**  
The multicast address is invalid, or the socket is not a member of the group.
- **NU\_MEM\_ALLOC**  
Insufficient memory.
- **NU\_INVALID\_PARM**  
mreq is invalid.

### Example

```
INT socket;  
struct IP6_MREQ mreq;  
  
/* Create a socket. */  
. . .  
  
/* Join the multicast group. */  
. . .  
  
/* Leave the multicast group. */  
NU_Setsockopt_IPV6_LEAVE_GROUP(socket, &mreq);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_MULTICAST\_HOPS (IPv6)

This function sets the hop limit to use in IPv6 multicast packets transmitted using this socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_MULTICAST_HOPS (INT socketd,  
                                           INT hop_limit);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **hop\_limit**  
The hop limit value to use in IPv6 multicast packets transmitted using this socket. By default, this value is set to `IP6_DEFAULT_MULTICAST_TTL` located in *networking/net6\_cfg.h*. If `hop_limit` is set to `-1`, the default value will be used. Otherwise, acceptable values are 0 – 255.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID**  
`hop_limit` is outside the allowable range.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the multicast structure for the socket.

### Example

```
INT    socket;  
INT    hop_limit = 1;  
  
/* Set the hop limit to use in multicast packets transmitted  
 * using this socket.  
 */  
NU_Setsockopt_IPV6_MULTICAST_HOPS(socket, hop_limit);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_MULTICAST\_IF (IPv6)

This function sets the interface to use for IPv6 multicast packets transmitted using this socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_MULTICAST_IF (INT    socketd,  
                                         INT32  if_index);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **if\_index**  
The index of the interface to use.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID**  
**if\_index** is invalid.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the multicast structure for the socket.

### Example

```
INT    socket;  
INT    if_index = 1;  
  
/* Set the interface to use for multicast packets transmitted  
 * using this socket.  
 */  
NU_Setsockopt_IPV6_MULTICAST_IF(socket, if_index);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_NEXTHOP (IPv6)

This function sets the Sticky Option for the Next-Hop to be used with all packets sent from this socket. This Sticky Option can be overridden on a per-packet basis with a call to NU\_Sendmsg that specifies a different Next-Hop as ancillary data.

### Usage

```
STATUS NU_Setsockopt_IPV6_NEXTHOP (INT          socketd,  
                                   struct addr_struct *address,  
                                   INT          length);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **address**  
A pointer to the Next-Hop to set as the Sticky Option. The [addr\\_struct](#) data structure is defined in the [Net Data Structures](#) section.
- **length**  
The length of the [addr\\_struct](#) structure. A value of zero will clear the Next-Hop Sticky Option for this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID**  
address is NULL and length is not zero.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket or the Sticky Option buffer is full.

### Example

```
INT          socket;  
struct addr_struct next_hop;  
  
/* Copy the Next-Hop address to use in the addr_struct data  
 * structure.  
 */  
memcpy(&next_hop.id, next_hop_address, IP6_ADDR_LEN);  
  
/* Set the Next-Hop Sticky Option to be used for all packets  
 * sent from this socket.
```



```

*/
NU_Setsockopt_IPV6_NEXTHOP(socket, &next_hop,
                           sizeof(struct addr_struct));

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>	<a href="#">NU_Getsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_IPV6\_PKTINFO (IPv6)

This function sets the source address and/or outgoing interface index Sticky Option on the socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_PKTINFO (INT          socketd,  
                                   struct in6_pktinfo *pkt_info);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **pkt\_info**  
A pointer to the data structure which contains the source address and/or interface index to set as Sticky Options for outgoing packets on the socket.  
The structure [in6\\_pktinfo](#) is defined in the [Net Data Structures](#) section.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVAL**  
The length of the options are greater than the stack can handle, the address specified does not exist on the node, the socket type is TCP and the source address cannot be changed, the interface index is invalid, the interface is not IPv6-enabled, or **pkt\_info** is NULL.

### Example

```
/* Set the source address and interface index to use for all packets sent  
from this socket. */  
  
INT      socket;  
in6_pktinfo pkt_info;  
  
/* Build the socket. */  
. . .  
  
/* Set the source address Sticky Option. This address will be  
* used in all IPv6 packets sent on this socket unless ancillary  
* data is specified to override it.  
*/
```

```
memcpy(pkt_info.ipi6_addr, source_address, IP6_ADDR_LEN);

/* Set the interface index Sticky Option. All packets sent on
 * this socket will be transmitted out of this interface unless
 * ancillary data is specified to override it.
 */
pkt_info.ipi6_ifindex = 1;

/* Set the socket option. */
NU_Setsockopt_IPV6_PKTINFO(socket, &pkt_info);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVDSTOPTS (IPv6)

This function enables or disables the value of the socket option to receive the Destination Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVDSTOPTS (INT socketd,  
                                       INT opt_val);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **opt\_val**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Example

```
/* Enable the socket option to receive the Destination Options of the most  
recently received packet as ancillary data from the stack. */  
  
INT  socket;  
  
/* Build the socket. */  
  
/* Indicate that the stack should return the Destination Options  
 * of the most recently received packet for this socket with the  
 * next call to NU_Recvmsg().  
 */  
NU_Setsockopt_IPV6_RECVDSTOPTS(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVHOPLIMIT (IPv6)

This function sets the socket option to return the hop limit of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVHOPLIMIT (INT socketd,  
                                         INT optval);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.

### Example

```
INT socket;  
  
/* Enable the socket option to return the Hop Limit of the most  
 * recently received packet on the socket with the next call to  
 * NU_Recvmsg().  
 */  
NU_Setsockopt_IPV6_RECVHOPLIMIT(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVHOPOPTS (IPv6)

This function enables or disables the value of the socket option to receive the Hop-By-Hop Options of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVHOPOPTS (INT socketd,  
                                       INT opt_val);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **opt\_val**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Example

```
/* Enable the socket option to receive the Hop-By-Hop Options of the most  
recently received packet as ancillary data from the stack. */  
  
INT  socket;  
  
/* Build the socket. */  
. . .  
  
/* Indicate that the stack should return the Hop-By-Hop Options  
* of the most recently received packet for this socket with the  
* next call to NU_Recvmsg().  
*/  
NU_Setsockopt_IPV6_RECVHOPOPTS(socket, 1);
```



## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVPKTINFO (IPv6)

This function sets the socket option to return the receive interface and/or destination address of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVPKTINFO (INT socketd,  
                                       INT optval);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Example

```
INT  socket;  
.  
/* Build the socket. */  
.  
.  
.  
  
/* Indicate that the stack should return the receive interface and  
 * destination address of the most recently received packet for  
 * this socket with the next call to NU_Recvmsg().  
 */  
NU_Setsockopt_IPV6_RECVPKTINFO(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVRTHDR (IPv6)

This function enables or disables the value of the socket option to receive the Routing Header of the most recently received packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVRTHDR (INT socketd,  
                                     INT opt_val);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **opt\_val**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Example

```
/* Enable the socket option to receive the Routing Header of the most  
recently received packet as ancillary data from the stack. */  
  
INT socket;  
  
/* Build the socket. */  
  
/* Indicate that the stack should return the Routing Header of  
 * the most recently received packet for this socket with the  
 * next call to NU_Recvmsg().  
 */  
NU_Setsockopt_IPV6_RECVRTHDR(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RECVTCLASS (IPv6)

This function sets the socket option that causes NU\_Recvmsg() to return the Traffic Class field from the IPv6 header of the incoming packet. This socket option cannot be used with TCP sockets since ancillary data cannot be received over TCP sockets.

### Usage

```
STATUS NU_Setsockopt_IPV6_RECVTCLASS (INT socketd,  
                                      INT optval);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.

### Example

```
INT socket;  
  
/* Set the socket option to return the Traffic Class of the most  
 * recently received packet on this socket with the next call to  
 * NU_Recvmsg().  
 */  
NU_Setsockopt_IPV6_RECVTCLASS(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RTHDR (IPv6)

This function sets the Routing Header Sticky Option for this socket. The Routing Header set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU\_Sendmsg specifying a different Routing Header as ancillary data.

### Usage

```
STATUS NU_Setsockopt_IPV6_RTHDR (INT          socketd,
                                struct ip6_rthdr *rt_hdr,
                                INT          length);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **rt\_hdr**  
A pointer to the IPv6 Routing Header built by the application. The [ip6\\_rthdr](#) data structure is defined in the [Net Data Structures](#) section. Note that this data structure is built by the application using the following API routines: [NU\\_Inet6\\_Rth\\_Space \(IPv6\)](#), [NU\\_Inet6\\_Rth\\_Init \(IPv6\)](#), and [NU\\_Inet6\\_Rth\\_Add \(IPv6\)](#).
- **length**  
The length of the `rt_hdr` structure. A value of zero disables the Routing Header Sticky Option on this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket, or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID**  
The [ip6\\_rthdr](#) data structure is `NU_NULL`, but the value of the length parameter is not zero.

### Example

```
INT      socket;
struct ip6_rthdr *ip6_rthdr;
UINT32   rt_space;

/** Build a Type 0 Routing Header with five Intermediate Nodes. */

/* Determine the amount of memory required for the Routing
 * Header and five intermediate nodes.
```



```

    */
    rt_space = NU_Inet6_Rth_space(IPV6_RTHDR_TYPE_0, 5);

    /* Allocate memory for the Routing Header. */
    NU_Allocate_Memory(&System_Memory, (VOID**)&ip6_rthdr, rt_space,
                      NU_NO_SUSPEND);

    /* Initialize the ip6_rthdr structure. */
    ip6_rthdr = NU_Inet6_Rth_Init(ip6_rthdr, rt_space,
                                IPV6_RTHDR_TYPE_0, 5);

    /* Add each intermediate node. */
    NU_Inet6_Rth_Add(ip6_rthdr, I1);
    NU_Inet6_Rth_Add(ip6_rthdr, I2);
    NU_Inet6_Rth_Add(ip6_rthdr, I3);
    NU_Inet6_Rth_Add(ip6_rthdr, I4);
    NU_Inet6_Rth_Add(ip6_rthdr, D);

    /* Set the Routing Header Sticky Option to use with all packets
     * sent on this socket.
     */
    NU_Setsockopt_IPV6_RTHDR(socket, ip6_rthdr, rt_space);

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_RTHDRDSTOPTS (IPv6)

This function sets the Destination Options extension header Sticky Option for the Destination Options to precede the Routing Header for this socket. The Destination Options set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU\_Sendmsg specifying a different Destination Options extension header as ancillary data.

### Usage

```
STATUS NU_Setsockopt_IPV6_RTHDRDSTOPTS (INT          socketd,  
                                         struct ip6_dest *dst_hdr,  
                                         INT          length);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **dst\_hdr**  
A pointer to the IPv6 Destination Options extension header built by the application. Note that this data structure is built by the application using the following API routines: [NU\\_Inet6\\_Opt\\_Init \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Append \(IPv6\)](#), [NU\\_Inet6\\_Opt\\_Finish \(IPv6\)](#), and [NU\\_Inet6\\_Opt\\_Set\\_Val \(IPv6\)](#).
- **length**  
The length of the ip6\_dest structure. A value of zero disables the Destination Options Sticky Option on this socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket, or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID**  
The value of the ip6\_dest structure is NU\_NULL, but the value of the length parameter is not zero.

### Example

```
INT      socket;  
INT      currentlen;  
UINT32   extlen;  
VOID     *databuf;  
INT      offset;
```

```

UINT8    value1;
UINT16   value2;
UINT32   value4;
UINT32   value8;
struct ip6_dest *dst_hdr;

/** Build Two Destination Options .**/

/* Determine the amount of memory for the Destination Options. */
currentlen = NU_Inet6_Opt_Init(NU_NULL, 0);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_X, 12,
                                8, NU_NULL);
currentlen = NU_Inet6_Opt_Append(NU_NULL, 0, currentlen, OPT_Y, 7,
                                4, NU_NULL);
currentlen = NU_Inet6_Opt_Finish(NU_NULL, 0, currentlen);

extlen = currentlen;

/* Allocate memory for the Destination Options. */
NU_Allocate_Memory(&System_Memory, (VOID**)&dst_hdr, extlen,
                  NU_NO_SUSPEND);

/* Initialize the Destination Options data structure. */
currentlen = NU_Inet6_Opt_Init(dst_hdr, extlen);

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_X, 12, 8, &databuf);

/* Insert value 0x12345678 for a 4-octet field. */
offset = 0;

value4 = 0x12345678;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof
                              (value4));

/* Insert value 0x0102030405060708 for an 8-octet field. */
value8 = 0x0102030405060708;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value8, sizeof
                              (value8));

currentlen = NU_Inet6_Opt_Append(dst_hdr, extlen, currentlen,
                                OPT_Y, 7, 4, &databuf);

/* Insert value 0x01 for an 1-octet field. */
offset = 0;

value1 = 0x01;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value1, sizeof
                              (value1));

/* Insert value 0x1331 for a 2-octet field. */
value2 = 0x1331;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value2, sizeof
                              (value2));

/* Insert value 0x01020304 for a 4-octet field. */
value4 = 0x01020304;
offset = NU_Inet6_Opt_Set_Val(databuf, offset, &value4, sizeof

```

```

        (value4));

currentlen = NU_Inet6_Opt_Finish(dst_hdr, extlen, currentlen);

/* Set the Destination Options Sticky Option to use with all
 * packets sent on this socket.
 */
NU_Setsockopt_IPV6_RTHDRDSTOPTS(socket, dst_hdr, currentlen);

/* Set the Routing Header Sticky Option. */
. . .

```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_TCLASS (IPv6)

This function sets the Traffic Class Sticky Option on the socket. The Traffic Class set will be used in all packets sent on this socket. This Sticky Option value can be overridden by a call to NU\_Sendmsg specifying a different Traffic Class as ancillary data.

### Usage

```
STATUS NU_Setsockopt_IPV6_TCLASS (INT socketd,  
                                INT tclass);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **tclass**  
The Traffic Class Sticky Option to set on the socket. A value of -1 clears the Traffic Class Sticky Option set on this socket. This value must be between -1 and 255.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_MEM\_ALLOC**  
Insufficient memory to create the Sticky Option buffer for the socket, or the Sticky Option buffer is full.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID\_PARM**  
The Traffic Class value is invalid.

### Example

```
INT    socket;  
  
/* Set the Traffic Class Sticky Option to use with all packets  
 * sent on this socket.  
 */  
NU_Setsockopt_IPV6_TCLASS(socket, 128);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_IPV6\_UNICAST\_HOPS (IPv6)

This function sets the value of the Hop Limit for the specified socket.

### Usage

```
STATUS NU_Setsockopt_IPV6_UNICAST_HOPS (INT    socketd,  
                                         INT16  hop_limit);
```

### Arguments

- **socketd**  
The socket for which to get the Hop Limit.
- **hop\_limit**  
The new hop limit of the socket. This value must be between -1 and 255. A value of -1 sets the Hop Limit of the socket to the default value, which is `IP6_HOP_LIMIT` by default, but can be changed by the Application Layer using the routine [NU\\_Set\\_Default\\_Hop\\_Limit \(IPv6\)](#).

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID\_PARM**  
One of the incoming parameters is invalid.

### Example

```
INT    socket;  
INT16  hop_limit = 128;  
  
/* Set the Hop Limit of this socket. */  
NU_Setsockopt_IPV6_UNICAST_HOPS(socket, hop_limit);
```

**Related Topics**

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	



## NU\_Setsockopt\_IPV6\_V6ONLY (IPv6)

This function sets the socket option for a socket of family type NU\_FAMILY\_IP6 to be used for IPv6 communications only.

### Usage

```
STATUS NU_Setsockopt_IPV6_V6ONLY (INT socketd,  
                                  INT optval);
```

### Arguments

- **socketd**  
The socket for which to set the option.
- **optval**  
A value of zero disables the option on the socket. A non-zero value enables the option on the socket. This option is disabled by default.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid. Either it is not a valid socket descriptor or the family type of the socket is not NU\_FAMILY\_IP6.

### Description

By setting this socket option, the socket can be used to send and receive IPv6 packets only; for example, traffic can be sent to and received from native IPv6 addresses only.

### Example

```
INT socket;  
  
/* Create a socket of family type NU_FAMILY_IP6. */  
. . .  
  
/* Set the socket option to transmit and receive IPv6 packets  
 * only on this socket.  
 */  
NU_Setsockopt_IPV6_V6ONLY(socket, 1);
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVDSOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_SO\_BROADCAST (IPv4)

This function enables or disables the broadcast status of a socket. It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `SOL_SOCKET` and type of `SO_BROADCAST`.

### Usage

```
STATUS NU_Setsockopt_SO_BROADCAST (INT  socketd,  
                                   INT16 optval);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
A value of zero disables broadcasts on the socket. A non-zero value enables broadcasts on the socket. Broadcasts are enabled on the socket by default.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.

### Example

```
INT socketd; /* socket descriptor */  
  
/* Build a socket.*/  
  
/* Enable broadcasts on the socket. */  
if (NU_Setsockopt_SO_BROADCAST(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>
<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	

## NU\_Setsockopt\_SO\_KEEPALIVE (IPv4/IPv6)

This function enables or disables TCP Keep-Alive on a socket. It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_TCP` and type of `SO_KEEPALIVE`.

### Usage

```
STATUS NU_Setsockopt_SO_KEEPALIVE (INT    socketd,  
                                   UINT8  opt_val);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `opt_val`  
A value of zero disables TCP Keep-Alive for TCP connections on the socket. A non-zero value enables TCP Keep-Alive for TCP connections on the socket. Keep-Alive is disabled by default.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.
- `NU_INVAL`  
There is no port structure associated with this socket.
- `NU_UNAVAILABLE`  
TCP Keep-Alive was not enabled at compile-time.

### Example

```
INT    socketd; /* socket descriptor */  
  
/* Build a TCP socket.*/  
  
/* Set the flag to enable TCP Keep-Alive on the socket. */  
if (NU_Setsockopt_SO_KEEPALIVE(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_SO\_LINGER (IPv4/IPv6)

This function configures the socket linger option. When this option is set and you close the socket, the close call will not return until one of the following actions occurs:

- All queued messages for the socket have been sent successfully.
- The linger timeout is reached.

This function performs the same functionality as a call to the function `NU_Setsockopt` with a level of `SOL_SOCKET` and type of `SO_LINGER`.

### Usage

```
STATUS NU_Setsockopt_SO_LINGER (INT socketd,  
                                struct sck_linger_struct optval);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `optval`  
The parameters of the socket linger option in the [sck\\_linger\\_struct](#) structure. A non-zero value in the `s_linger.linger_on` parameter indicates that the `SO_LINGER` option should be enabled on the socket, and the `s_linger.linger_ticks` parameter contains the number of ticks the socket will linger before closing.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.

### Example

```
INT socketd; /* socket descriptor */  
struct sck_linger_struct optval;  
  
/* Build a socket.*/  
  
/* Configure the socket linger option. */  
optval.linger_on = NU_TRUE;  
  
/* Suspend until the close completes. */  
optval.linger_ticks = NU_SUSPEND;  
  
if (NU_Setsockopt_SO_LINGER(socketd, optval) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>



## NU\_Setsockopt\_SO\_RCVBUF (IPv4/IPv6)

This function sets the TCP Window Size of the local side of the connection for a specified socket. This routine has been deprecated in favor of

[NU\\_Setsockopt\\_TCP\\_RCV\\_WINDOWSIZE](#) in order to take advantage of the expanded Window Size field enabled by using the TCP Window Scale option.

### Usage

```
STATUS NU_Setsockopt_SO_RCVBUF (INT    socketd,  
                                INT32  opt_val);
```

### Arguments

- **socketd**  
The TCP socket descriptor for which to configure the local TCP Window Size.
- **opt\_val**  
The local Window Size to set for this socket. The value must be greater than zero. The default value for all new sockets is `WINDOW_SIZE`, set to 16000 by default in *networking/net\_cfg.h*.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The specified socket descriptor is invalid.
- **NU\_INVALID\_PARM**  
The specified window size is invalid.

### Description

This function must be called before initiating a connection via `NU_Connect` or before marking the socket as a listening socket via `NU_Listen`. When this option is set for a listening socket, all new connections created from the listening socket will inherit the TCP Window Size of the listening socket.

### Example

```
INT    socketd;  
struct addr_structservaddr;  
  
/* Create a TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
/* Reduce the local TCP Window size of the socket to 12000. */  
status = NU_Setsockopt_SO_RCVBUF(socketd, 12000);  
  
if (status != NU_SUCCESS)  
{
```

```
        printf("Error at call to NU_Setsockopt_SO_RCVBUF.\n");
    }

    /* Fill in the addr_struct. */
    . . .

    /* Initiate a connection using the TCP socket. */
    if (NU_Connect(socketd, &servaddr, 0) >= 0)
    {
        . . .
    }
```

## Related Topics

[Net API Functions](#)[NU\\_Getsockopt\\_SO\\_RCVBUF \(IPv4/IPv6\)](#)[NU\\_Setsockopt \(IPv4/IPv6\)](#)[NU\\_Getsockopt \(IPv4/IPv6\)](#)[Net API Functions](#)

## NU\_Setsockopt\_SO\_REUSEADDR (IPv4/IPv6)

This function configures the reuse address option.

### Usage

```
STATUS NU_Setsockopt_SO_REUSEADDR (INT    socketd,  
                                   INT16  optval);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **optval**  
Specifies the new value of the SO\_REUSEADDR option. A value of zero disables the option. A non-zero value enables the option on the socket.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

### Description

It performs the same functionality as a call to the function NU\_Setsockopt with a level of SOL\_SOCKET and type of SO\_REUSEADDR. This option allows multiple addresses to bind to the same port number. Note, however, that the same address cannot bind to the same port number. The addresses must be unique.

### Example

```
INT    socketd; /* socket descriptor */  
INT16  optval = 1;  
  
/* Build a socket.*/  
  
/* Set the reuse address option. */  
if (NU_Setsockopt_SO_REUSEADDR(socketd, optval) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_TCP\_CFG\_DSACK

This function toggles Duplicate Selective Acknowledgement (D-SACK) support on a TCP socket at run-time. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the D-SACK value of the parent socket. This option can be toggled on a socket any time before or after an active connection is established.

### Usage

```
STATUS NU_Setsockopt_TCP_CFG_DSACK (INT    socketd,  
                                     UINT8  opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the D-SACK option for the socket. A value of zero disables D-SACK on the socket. A non-zero value enables D-SACK on the socket. If D-SACK is enabled in the system at compile-time, D-SACK is enabled on all new sockets by default.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
SACK has been disabled on the socket; therefore, D-SACK is also disabled.
- **NU\_UNAVAILABLE**  
D-SACK support was not compiled into the system; therefore, this option is not configurable on the socket.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */
```

```
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Disable D-SACK for the socket. */
    status = NU_Setsockopt_TCP_CFG_DSACK(socketd, 0);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_CFG\_SACK

This function toggles Selective Acknowledgement (SACK) support on a TCP socket at run-time. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the SACK value of the parent socket. This option can be toggled only before a connection has been accepted on the socket.

### Usage

```
STATUS NU_Setsockopt_TCP_CFG_SACK (INT    socketd,  
                                   UINT8  opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the SACK option for the socket. A value of zero disables SACK on the socket. A non-zero value enables SACK on the socket. If SACK is enabled in the system at compile-time, SACK is enabled on all new sockets by default.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_UNAVAILABLE**  
SACK support was not compiled into the system; therefore, this option is not configurable on the socket.
- **NU\_INVALID**  
A connection has already been established on the socket, and this option is no longer configurable.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .
```

```
/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Disable SACK for the socket. */
    status = NU_Setsockopt_TCP_CFG_SACK(socketd, 0);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_DSACK](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)



## NU\_Setsockopt\_TCP\_CONGESTION\_CTRL

This function toggles support for the entire set of TCP Congestion Control algorithms supported by the Nucleus NET software on a TCP socket at run-time. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the congestion control value of the parent socket. This option can be toggled only before a connection has been accepted on the socket.

### Usage

```
STATUS NU_Setsockopt_TCP_CONGESTION_CTRL(INT    socketd,  
                                         UINT8  opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the TCP Congestion Control algorithm option for the socket. A value of zero disables all congestion control algorithms on the socket. A non-zero value enables congestion control on the socket. If congestion control is enabled in the system at compile-time, congestion control is enabled on all new sockets by default.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_UNAVAILABLE**  
Congestion control support was not compiled into the system; therefore, this option is not configurable on the socket.
- **NU\_INVALID**  
A connection has already been established on the socket, and this option is no longer configurable.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
. . .
```

```
/* Initialize each networking interface. */
. . .

/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Disable Congestion Control for the socket. */
    status = NU_Setsockopt_TCP_CONGESTION_CTRL(socketd, 0);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_FIRST\\_PROBE\\_TIME](#)   [NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)  
[OUT](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)   [NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_DELAY\_ACK

This function configures the amount of time to delay TCP ACKs on a TCP socket at run-time when less than two full-size segments have been received to trigger an ACK. This timeout value is set to TCP\_ACK\_TIMEOUT by default when a new TCP socket is created.

TCP\_ACK\_TIMEOUT can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_DELAY_ACK(INT      socketd,  
                                   UINT32  delay);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **delay**  
Value in hardware ticks to delay transmitting ACKs for incoming data for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to delay ACKing incoming data segments  
     * by one second.  
     */  
}
```

```
status =
    NU_Setsockopt_TCP_DELAY_ACK(socketd,
                                1 * SCK_Ticks_Per_Second);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_FIRST\_PROBE\_TIMEOUT

This function configures the timeout value for sending the initial Zero Window probe on a TCP socket at run-time. This timeout value is set to PROBE\_TIMEOUT by default when a new TCP socket is created. PROBE\_TIMEOUT can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept() during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_FIRST_PROBE_TIMEOUT(INT socketd,  
                                              INT32 timeout);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **timeout**  
Value in hardware ticks to delay before sending the initial Zero Window probe on the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The value of the timeout parameter is not a positive integer.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to delay five seconds before transmitting
```

```
    * the first Zero Window probe.
    */
    status =
        NU_Setsockopt_TCP_FIRST_PROBE_TIMEOUT(socketd,
                                              5 * SCK_Ticks_Per_Second);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_FIRST\_RTO

This function configures the value of the first retransmission timeout on a TCP socket at run-time. This timeout value is set to TCP\_RTTDFLT by default when a new TCP socket is created. TCP\_RTTDFLT can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. The value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_FIRST_RTO(INT  socketd,
                                   INT32 timeout);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **timeout**  
Value in hardware ticks to delay before sending the first retransmission of an unACKed data packet for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The value of the timeout parameter is not a positive integer.

### Example

```
INT      socketd;
STATUS   status;

/* Initialize Nucleus NET. */
. . .

/* Initialize each networking interface. */
. . .

/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Configure the socket to wait five seconds before sending
```

```
    * the first retransmission of a data segment.
    */
    status =
        NU_Setsockopt_TCP_FIRST_RTO(socketd,
                                     5 * SCK_Ticks_Per_Second);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)



## NU\_Setsockopt\_TCP\_KEEPALIVE\_R2

This function configures the maximum number of unanswered keep-alive probes to transmit on a TCP socket at run-time. This value is set to TCP\_MAX\_KEEPALIVES by default when a new TCP socket is created. TCP\_MAX\_KEEPALIVES can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_KEEPALIVE_R2 (INT    socketd,  
                                       UINT8  max_retrans);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **max\_retrans**  
Maximum number of retransmissions of unanswered Keep-Alive packets for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The input parameter max\_retrans is set to zero.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to retransmit five Keep-Alive packets
```

```
    * before closing the connection.
    */
    status = NU_Setsockopt_TCP_KEEPALIVE_R2(socketd, 5);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_KEEPALIVE\_WAIT

This function configures the amount of time to remain idle on a TCP socket before invoking the TCP keep-alive protocol at run-time. This value is set to TCP\_KEEPALIVE\_INTERVAL by default when a new TCP socket is created. TCP\_KEEPALIVE\_INTERVAL can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_KEEPALIVE_WAIT(INT    socketd,  
                                         UINT32 delay);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **delay**  
Value in hardware ticks to allow a connection to remain idle before invoking the keep-alive protocol.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The delay parameter is set to zero.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{
```

```
/* Configure the socket to invoke the Keep-Alive mechanism after
 * 30 minutes of inactivity on the connection.
 */
status =
    NU_Setsockopt_TCP_KEEPALIVE_WAIT(socketd,
                                     30 * 60 * SCK_Ticks_Per_Second);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_R2](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_MAX\_PROBES

This function configures the maximum number of Zero Window probes to be transmitted before giving up on window probing on a TCP socket at run-time. This value is set to TCP\_MAX\_PROBE\_COUNT by default when a new TCP socket is created. TCP\_MAX\_PROBE\_COUNT can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_MAX_PROBES(INT  socketd,  
                                     UINT8 max_probes);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **max\_probes**  
The maximum number of unanswered Zero Window probes to transmit over the specified socket before closing the connection.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to transmit a maximum of seven Zero Window  
     * probes before giving up on the connection.  
     */  
}
```

```
status = NU_Setsockopt_TCP_MAX_PROBES(socketd, 7);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_R2](#)

## NU\_Setsockopt\_TCP\_MAX\_R2

This function configures the number of times to retransmit a data segment before closing the connection on a TCP socket at run-time. This value is set to MAX\_RETRANSMITS by default when a new TCP socket is created. MAX\_RETRANSMITS can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_MAX_R2 (INT  socketd,  
                                INT8 max_retrans);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **max\_retrans**  
Maximum number of retransmissions of an unACKed data packet for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
...  
  
/* Initialize each networking interface. */  
...  
  
/* Create a new TCP socket. */  
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to retransmit a data segment a maximum  
     * of five times before closing the connection.  
     */  
}
```

```
status = NU_Setsockopt_TCP_MAX_R2 (socketd, 5);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)



## NU\_Setsockopt\_TCP\_MAX\_RTO

This function configures the maximum retransmission timeout value on a TCP socket at run-time. This timeout value is set to MAXRTO by default when a new TCP socket is created. MAXRTO can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_MAX_RTO(INT    socketd,  
                                UINT32 timeout);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **timeout**  
Maximum value in hardware ticks to delay between successive retransmissions of an unACKed data packet for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The value of the timeout parameter is set to zero.

### Example

```
INT    socketd;  
STATUS status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the socket to use a maximum retransmission timeout
```

```
    * of seven seconds on the socket.
    */
    status =
        NU_Setsockopt_TCP_MAX_RTO(socketd,
                                   7 * SCK_Ticks_Per_Second);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_MAX\_SYN\_R2

This function configures the number of times to retransmit a SYN packet before closing the connection on a TCP socket at run-time. This timeout value is set to MAX\_RETRANSMITS by default when a new TCP socket is created. MAX\_RETRANSMITS can be configured at compile-time in the file *net\_cfg.h*. Note that a connection must not have been made on the socket before invoking this routine.

### Usage

```
STATUS NU_Setsockopt_TCP_MAX_SYN_R2 (INT    socketd,
                                     UINT8  max_retrans);
```

### Arguments

- **socketd**  
 Socket for which to configure the value.
- **max\_retrans**  
 Maximum number of retransmissions of an unACKed SYN packet for the specified socket before the connection is closed.

### Return Values

- **NU\_SUCCESS**  
 Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
 The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
 The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
 A connection has already been established on this socket; therefore this value cannot be set.

### Example

```
INT          socketd;
STATUS       status;

/* Initialize Nucleus NET. */
. . .

/* Initialize each networking interface. */
. . .

/* Create a new TCP socket. */
socketd = NU_Socket (NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Configure the socket to retransmit a SYN packet a maximum
```

```
    * of three times before giving up on the new connection.
    */
    status = NU_Setsockopt_TCP_MAX_SYN_R2(socketd, 3);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_RTO](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_MSL

This function configures the amount of time to wait before reusing a closed port / IP address combination on a TCP socket at run-time. This timeout value is set to WAITTIME by default when a new TCP socket is created. WAITTIME can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_MSL(INT      socketd,  
                             UINT32  msl);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **msl**  
The value in hardware ticks to set for the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{  
    /* Configure the system to wait 10 seconds before allowing a new  
     * socket to reuse the port / IP address combination set on this  
     * socket.  
     */  
    status =  
        NU_Setsockopt_TCP_MSL(socketd,
```

```
                                10 * SCK_Ticks_Per_Second);  
  
    if (status == NU_SUCCESS)  
    {  
        . . .  
    }  
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_KEEPALIVE\\_WAIT](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_PROBE\_TIMEOUT

This function configures the maximum timeout value for sending successive Zero Window probes on a TCP socket at run-time. This timeout value is set to MAX\_PROBETIMEOUT by default when a new TCP socket is created. MAX\_PROBETIMEOUT can be configured at compile-time in the file *net\_cfg.h*. The configured value is inherited by all child sockets created from the parent socket at the call to NU\_Accept() during run-time. This value can be configured at any time.

### Usage

```
STATUS NU_Setsockopt_TCP_PROBE_TIMEOUT(INT socketd,  
                                       INT32 timeout);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **timeout**  
Value in hardware ticks to delay between successive Zero Window probes on the specified socket.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
The value of the timeout parameter is not a positive integer.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .  
  
/* Create a new TCP socket. */  
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);  
  
if (socketd >= 0)  
{
```

```
/* Configure the socket to delay two seconds between consecutive
 * Zero Window probes.
 */
status =
    NU_Setsockopt_TCP_PROBE_TIMEOUT(socketd,
                                     2 * SCK_Ticks_Per_Second);

if (status == NU_SUCCESS)
{
    . . .
}
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)



## NU\_Setsockopt\_TCP\_NODELAY (IPv4/IPv6)

This function enables or disables the value of the Nagle Algorithm. It performs the same functionality as a call to the function `NU_Setsockopt` with a level of `IPPROTO_TCP` and type of `TCP_NODELAY`.

### Usage

```
STATUS NU_Setsockopt_TCP_NODELAY (INT    socketd,  
                                  UINT8  opt_val);
```

### Arguments

- `socketd`  
Specifies a socket descriptor.
- `opt_val`  
A non-zero value disables the Nagle Algorithm on the socket; that is, eliminates the delay in transmission. A value of zero enables the Nagle Algorithm on the socket.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- `NU_NOT_CONNECTED`  
A connection does not exist on the specified socket.
- `NU_INVALID`  
Invalid parameter.

### Example

```
INT socketd; /* socket descriptor */  
  
/* Build a TCP socket.*/  
  
/* Disable the Nagle Algorithm on the socket. */  
if (NU_Setsockopt_TCP_NODELAY(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_UDP_NOCHECKSUM (IPv4)</a>	<a href="#">NU_Setsockopt (IPv4/IPv6)</a>

## NU\_Setsockopt\_TCP\_RCV\_WINDOWSIZE

This function configures the size of the local receive window for a socket and computes the subsequent shift count required to fit the new window size into the 16-bit field of the TCP header when the TCP Window Scale option is enabled. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the receive window size and shift count of the parent socket. This option can be configured only before a connection has been established.

### Usage

```
STATUS NU_Setsockopt_TCP_RCV_WINDOWSIZE (INT    socketd,  
                                         UINT32 opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the local receive window for the socket. By default, this value is set to `WINDOW_SIZE` upon creation of a new socket. The maximum allowable window size for a socket is `TCP_MAX_WINDOW_SIZE` when the Window Scale option is enabled and 65535 when the Window Scale option is disabled. This value must be greater than zero.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to `NU_Socket`.
- **NU\_INVAL**  
A connection has already been made on the socket, or the value of the window size is 0 or greater than `TCP_MAX_WINDOW_SIZE`, or the value is greater than 65535 and the Window Scale option is disabled.

### Example

```
INT      socketd;  
STATUS   status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .
```

```
/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Configure the window size to be the maximum allowable for the
     * socket.
     */
    status = NU_Setsockopt_TCP_RCV_WINDOWSIZE(socketd, 1073741824);

    if (status == NU_SUCCESS)
    {
        /* Configure the scale factor to account for the large
         * window.
         */
        status = NU_Setsockopt_TCP_WINDOWSCALE(socketd, 14);

        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_TCP\_TIMESTAMP

This function toggles support for the TCP Timestamp option on a TCP socket at run-time. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the configured value of the parent socket. This option can be toggled only before a connection has been accepted on the socket.

### Usage

```
STATUS NU_Setsockopt_TCP_TIMESTAMP(INT    socketd,  
                                   UINT8  opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the Timestamp option for the socket. A value of zero disables the Timestamp option on the socket. A non-zero value enables the Timestamp option on the socket. If the Timestamp option is enabled in the system at compile-time, the option is enabled on all new sockets by default.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_UNAVAILABLE**  
Timestamp option support was not compiled into the system; therefore, this option is not configurable on the socket.
- **NU\_INVAL**  
A connection has already been made on the socket.

### Example

```
INT          socketd;  
STATUS       status;  
  
/* Initialize Nucleus NET. */  
. . .  
  
/* Initialize each networking interface. */  
. . .
```

```
/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Configure the socket to not use the Timestamp option. */
    status = NU_Setsockopt_TCP_TIMESTAMP(socketd, 0);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

## Related Topics

[Net API Functions](#)[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)[NU\\_Setsockopt\\_TCP\\_WINDOWSCALE](#)[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setsockopt\_UDP\_NOCHECKSUM (IPv4)

This function enables or disables the computation of the UDP checksum for outgoing IPv4 packets and the verification of the UDP checksum for incoming IPv4 packets transmitted and received using this socket. When this option is enabled, the value of the checksum for outgoing packets will be zero in the UDP header.

### Usage

```
STATUS NU_Setsockopt_UDP_NOCHECKSUM (INT    socketd,  
                                     UINT8  opt_val);
```

### Arguments

- socketd

Specifies a socket descriptor.

- opt\_val

A value of zero disables the UDP\_NOCHECKSUM option on the socket; enabling the computation and verification of the UDP checksum for the socket. A non-zero value enables the UDP\_NOCHECKSUM option on the socket; disabling the computation and verification of the UDP checksum for the socket. The option is disabled by default.

### Return Values

- NU\_SUCCESS

Upon successful completion.

- NU\_INVALID\_SOCKET

The socket parameter was not a valid socket value, or it had not been previously allocated via the NU\_Socket call.

- NU\_NOT\_CONNECTED

The socket has been closed.

### Example

```
INT socketd; /* socket descriptor */  
  
/* Build a UDP socket.*/  
  
/* Enable the UDP_NOCHECKSUM option on the socket. */  
if (NU_Setsockopt_UDP_NOCHECKSUM(socketd, 1) != NU_SUCCESS)  
{  
    /* Fatal error, abort here. */  
}
```

## Related Topics

<a href="#">NU_Setsockopt (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_IP_ADD_MEMBERSHIP (IPv4)</a>
<a href="#">NU_Setsockopt_IP_DROP_MEMBERSHIP (IPv4)</a>	<a href="#">NU_Setsockopt_IP_BROADCAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_HDRINC (IPv4)</a>	<a href="#">NU_Setsockopt_IP_MULTICAST_IF (IPv4)</a>
<a href="#">NU_Setsockopt_IP_MULTICAST_TTL (IPv4)</a>	<a href="#">NU_Setsockopt_IP_RECVIFADDR (IPv4)</a>
<a href="#">NU_Setsockopt_IP_TOS (IPv4)</a>	<a href="#">NU_Setsockopt_IP_TTL (IPv4)</a>
<a href="#">NU_Setsockopt_IPV6_CHECKSUM (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_DSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_HOPOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_JOIN_GROUP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_LEAVE_GROUP (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_MULTICAST_HOPS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_MULTICAST_IF (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_NEXTHOP (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_PKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVDSTOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVHOPLIMIT (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVHOPOPTS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVPKTINFO (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RECVRTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RECVTCLASS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_RTHDR (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_RTHDRDSTOPTS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_TCLASS (IPv6)</a>
<a href="#">NU_Setsockopt_IPV6_UNICAST_HOPS (IPv6)</a>	<a href="#">NU_Setsockopt_IPV6_V6ONLY (IPv6)</a>
<a href="#">NU_Setsockopt_SO_BROADCAST (IPv4)</a>	<a href="#">NU_Setsockopt_SO_KEEPALIVE (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_SO_LINGER (IPv4/IPv6)</a>	<a href="#">NU_Setsockopt_SO_REUSEADDR (IPv4/IPv6)</a>
<a href="#">NU_Setsockopt_TCP_NODELAY (IPv4/IPv6)</a>	



## NU\_Setsockopt\_TCP\_WINDOWSCALE

This function toggles support for the TCP Window Scale option on a TCP socket at run-time and sets the shift count for the local receive window per RFC 1323.

### Usage

```
STATUS NU_Setsockopt_TCP_WINDOWSCALE(INT socketd,  
                                     INT opt_val);
```

### Arguments

- **socketd**  
Socket for which to configure the value.
- **opt\_val**  
Value of the shift count for the socket. A negative value disables the TCP Window Scale option on the socket. A positive value enables the TCP Window Scale option on the socket and configures the shift count for the socket to that value. If the TCP Window Scale option is enabled in the system at compile-time, the option is enabled on all new sockets by default with a shift count equal to the smallest value that will allow the window size to fit into a 16-bit field. The maximum allowable shift count for a socket is TCP\_MAX\_WINDOWSCALE\_FACTOR.

### Return Values

- **NU\_SUCCESS**  
Indicates successful completion of the service.
- **NU\_INVALID\_SOCKET**  
The socket passed in does not reference a valid TCP socket descriptor.
- **NU\_NOT\_CONNECTED**  
The socket passed in has not been created via a call to NU\_Socket.
- **NU\_INVALID**  
A connection has already been made on the socket, or the value of the shift count is invalid; that is, greater than TCP\_MAX\_WINDOWSCALE\_FACTOR or less than the value required to fit the window size into a 16-bit field.
- **NU\_UNAVAILABLE**  
TCP Window Scale support was not compiled into the system; therefore, this option is not configurable on the socket.

### Description

The shift count that NU\_Setsockopt\_TCP\_WINDOWSCALE sets is used to extend the 16-bit advertised window field in the TCP header to a maximum of TCP\_MAX\_WINDOW\_SIZE. The configured shift count is included in the TCP Window Scale option of the outgoing SYN packet sent over the socket to indicate to the receiver the number of bits to left shift the

advertised 16-bit window. If the socket is a listening socket, any child sockets created from new connections received on this socket will inherit the respective values of the parent socket. This option can be configured only before a connection has been established. By default, the value is set to zero to indicate no shifting.

If support for the Window Scale option is included in the TCP/IP stack at compile-time, the shift count for the default window size will be automatically configured upon creation of a socket, or the window size is configured with the smallest shift count value that will allow the window size to fit within a 16-bit field. This routine should be used to disable the Window Scale option or set the window scale option to a value other than the default value configured by the stack.

If support for the Window Scale option is disabled on the socket, the local window size for the socket will revert to the default window size as configured by the `WINDOW_SIZE` macro in *net\_cfg.h*. If that value is greater than 65,535, the value for the socket will be set to 65,535.

### Example

```
INT          socketd;
STATUS       status;

/* Initialize Nucleus NET. */
. . .

/* Initialize each networking interface. */
. . .

/* Create a new TCP socket. */
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

if (socketd >= 0)
{
    /* Disable the Window Scale option for the socket. */
    status = NU_Setsockopt_TCP_WINDOWSCALE(socketd, -1);

    if (status == NU_SUCCESS)
    {
        . . .
    }
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Setsockopt\\_TCP\\_CFG\\_SACK](#)

[NU\\_Setsockopt\\_TCP\\_CONGESTION\\_CTRL](#)

[NU\\_Setsockopt\\_TCP\\_RCV\\_WINDOWSIZE](#)

[NU\\_Setsockopt\\_TCP\\_TIMESTAMP](#)

[NU\\_Setsockopt\\_TCP\\_MAX\\_PROBES](#)

## NU\_Setup\_Configured\_Tunnel (IPv6)

This function is responsible for setting up the IPv4 tunnel source and endpoint addresses of the configured tunnel for the IPv6 virtual device specified.

### Usage

```
STATUS NU_Setup_Configured_Tunnel (UINT8 *dev_name,  
                                  UINT8 *source_addr,  
                                  UINT8 *tunnel_endpoint);
```

### Arguments

- **dev\_name**  
Pointer to the name of the virtual device to configure.
- **source\_addr**  
Pointer to the IPv4 source address of the configured tunnel.
- **tunnel\_endpoint**  
Pointer to the the IPv4 endpoint address of the configured tunnel.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID\_PARM**  
One of the three parameters is NULL.

### Description

A configured tunnel is used to tunnel IPv6 packets in IPv4 packets to a pre-defined destination endpoint. The IPv6 packet is decapsulated at the tunnel endpoint and forwarded via IPv6 to the appropriate IPv6 node.

### Example

```
/* The IPv6 address assigned by the network administrator of the  
 * configured tunnel endpoint  
 */  
UINT8 Configured_Addr[] =  
{0x3f, 0xfe, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0xff, 0xfe, 0, 0};  
  
/* The IPv4 address of the tunnel endpoint */  
UINT8 Tunnel_Endpoint[] = {192, 168, 1, 1};  
  
/* The IPv4 address of the tunnel source */  
UINT8 Local_Ip[] = {192, 200, 100, 1};  
  
/* Initialize the network. */  
  
/* Set up the IPv4 device. */
```

```
/* Set up the IPv6 virtual device. */
DEMOI_Devices[1].dv_name = "Config1";
DEMOI_Devices[1].dv_init = CFG6_Initialize_Tunnel;
DEMOI_Devices[1].dv_flags = (DV6_IPV6 | DV6_PRIMARY_INT);
/* Initialize the two devices */
NU_Init_Devices(DEMOI_Devices, 2);

/* Add the globally unique IPv6 address assigned by the tunnel
 * endpoint administrator to the IPv6 virtual device.
 */
NU_Add_IP_To_Device(DEMOI_Devices[1].dv_name, Configured_Addr,
                    8, 0xffffffff, 0xffffffff);

/* Set up the configured virtual tunnel. */
NU_Setup_Configured_Tunnel(DEMOI_Devices[1].dv_name,
                           Local_Ip, Tunnel_Endpoint);
```

## Related Topics

[Net API Functions](#)

[NU\\_Socket\\_Connected](#)

## NU\_Socket\_Connected

This function returns NU\_TRUE if the specified socket has a foreign connection and NU\_FALSE if it does not.

### Usage

```
STATUS NU_Socket_Connected(INT socketd);
```

### Arguments

- `socketd`  
This is the socket descriptor.

### Return Values

- `NU_TRUE`  
The socket is connected.
- `NU_FALSE`  
The socket is not connected.
- `NU_INVALID_SOCKET`  
The socket descriptor number is outside the valid range of sockets that could have been opened.
- `NU_NOT_CONNECTED`  
The socket was not created with `NU_Socket`.

### Example

```
INT          socketd;

/* Open a TCP socket. */
. . .

/* Initiate a connection. */
. . .

/* Check if the socket is connected. */
if (NU_Socket_Connected(socketd))
{
    . . .
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Connect \(IPv4/IPv6\)](#)

## NU\_Set\_MDNS\_Hostname\_Callback

This API routine sets the callback routine that will be used to create host names for interfaces used in mDNS upon registration of a new interface or upon conflict detection.

### Usage

```
STATUS NU_Set_MDNS_Hostname_Callback (CHAR* (set_hostname (UINT32 int_index,  
                                                         BOOLEAN conflict)));
```

### Arguments

- `set_hostname`  
Pointer to the callback routine to invoke whenever the mDNS module needs to create a host name for a registered interface. Within the callback routine:
  - `int_index`  
The index of the interface for which the host name is being created.
  - `conflict`
    - `NU_TRUE` if a conflict was detected.
    - `NU_FALSE` otherwise.

### Return Values

- `NU_SUCCESS`  
The function completed successfully
- `NU_INVALID_PARM`  
The input parameter is invalid.
- Operating-system specific error

### Description

Note that the routine registered to set the host name must use dynamically allocated memory when returning the host name to the caller. This memory will be used for the new host name and must not be deallocated by the application. The mDNS module will deallocate the memory when the interface is removed or the corresponding interface records deleted.

If the application does not set a callback routine, [NU\\_Set\\_MDNS\\_Hostname\\_Callback](#) defaults to an internal routine, which appends a random number to the existing host name to create a new name.

### Examples

```
STATUS status;  
status = NU_Set_MDNS_Hostname_Callback (set_hostname);
```

```
/* Set_hostname routine */
CHAR* set_hostname(UINT32 int_index, BOOLEAN conflict)
{
    STATUS status;
    CHAR    *new_name = NU_NULL;
    CHAR    name[] = "mentor-interface.local";

    status = NU_Allocate_Memory(MEM_Cached, (VOID**)&new_name,
                                strlen(name) + 1, NU_SUSPEND);

    if (status == NU_SUCCESS)
    {
        strcpy(new_name, name);
    }

    return (new_name);
}
```

## Related Topics

[Net API Functions](#)

## NU\_Start\_mDNS\_Query

This routine invokes an mDNS continuous query for the indicated record type. This routine returns immediately, and the query operation occurs in the background in a separate thread. When an answer is received from a foreign node, the caller of this routine is signaled with a Nucleus PLUS signal.

### Usage

```
STATUS NU_Start_mDNS_Query(CHAR  *data,  
                           INT    type,  
                           INT16  family,  
                           VOID  *ptr);
```

### Arguments

- data  
Pointer to the data to use in the question part of the outgoing DNS query.
- type  
The type of record to query. The following record types are supported:
  - DNS\_TYPE\_A
  - DNS\_TYPE\_AAAA
  - DNS\_TYPE\_PTR
  - DNS\_TYPE\_SRV
  - DNS\_TYPE\_TXT
- family  
The family of the target record:
  - NU\_FAMILY\_IP
  - NU\_FAMILY\_IP6
  - NU\_FAMILY\_UNSPECif the record does not resolve to an IP address.
- ptr  
Currently unused. Intended for future functionality as required.

### Return Values

- NU\_SUCCESS  
The function completed successfully. The query has been started, and the calling thread will be signaled when a new record for the query is found.
- NU\_INVALID\_PARM  
An input parameter is invalid.



- An operating system specific error is returned otherwise.

## Example

```
VOID Start_Query(void)
{
    NU_HOSTENT  *host_ptr;
    STATUS      status;

    NU_Register_Signal_Handler(SH_Control);
    NU_Control_Signals(1UL << MDNS_SIGNAL);

    /* Query for A record - notify the application when changes are made to
     * the record in question.
     */
    status = NU_Start_mDNS_Query("www.mentor.com", DNS_TYPE_A,
                                NU_FAMILY_IP, NU_NULL);

    for (;;)
    {
        while (Application_Sleep == NU_TRUE)
            NU_Sleep(SCK_Ticks_Per_Second);

        /* Get the updated host record. */
        host_ptr = NU_Get_IP_Node_By_Name("mentor.com", NU_FAMILY_IP,
                                           (DNS_ALL | DNS_V4MAPPED), &status);

        /* Do something with the record. */
        . . .

        /* Free the memory for the record. */
        NU_Free_Host_Entry(host_ptr);

        /* Go back to sleep. */
        Application_Sleep == NU_TRUE;
    }
}

VOID SH_Control(UNSIGNED signals)
{
    if (signals & (1UL << MDNS_SIGNAL))
    {
        Application_Sleep = NU_FALSE;
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_Stop\\_mDNS\\_Query](#)

## NU\_Stop\_mDNS\_Query

This routine stops an mDNS continuous query that was initiated previously via a call to [NU\\_Start\\_mDNS\\_Query](#).

### Usage

```
STATUS NU_Stop_mDNS_Query(CHAR    *data,  
                           INT     type,  
                           INT16   family,  
                           VOID    *ptr);
```

### Arguments

- data  
Pointer to the data to use in the question part of the outgoing DNS query.
- type  
The type of record to query. The following record types are supported:  
DNS\_TYPE\_A  
DNS\_TYPE\_AAAA  
DNS\_TYPE\_PTR  
DNS\_TYPE\_SRV  
DNS\_TYPE\_TXT
- family  
The family of the record being queried:  
NU\_FAMILY\_IP  
NU\_FAMILY\_IP6  
NU\_FAMILY\_UNSPEC  
if the record does not resolve to an IP address.
- ptr  
Currently unused. Intended for future functionality as required.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the query has been stopped.
- NU\_INVALID\_PARM  
An input parameter is invalid.
- NU\_NO\_ACTION  
The caller is not currently querying the record.

- Operating system specific error code, otherwise.

### Example

```
/* Query for A record - notify the application when changes are made
 * to the record in question.
 */
status = NU_Start_mDNS_Query("www.mentor.com", DNS_TYPE_A,
                             NU_FAMILY_IP, NU_NULL);

. . .

/* Stop the query. */
status = NU_Stop_mDNS_Query("www.mentor.com", DNS_TYPE_A,
                             NU_FAMILY_IP, NU_NULL);
```

### Related Topics

[Net API Functions](#)

[NU\\_Start\\_mDNS\\_Query](#)

## NU\_TFTPGet2 (IPv4/IPv6)

Gets a file from a remote TFTP server.

### Usage

```
INT32 NU_TFTPGet2 (const UINT    *remote_ip,  
                  const CHAR    *rpath,  
                  CHAR          *lpath,  
                  TFTP_OPTIONS *ops,  
                  INT16         NU_FAMILY_IP);
```

### Arguments

- **remote\_ip**  
Pointer to the remote TFTP server IP address.
- **rpath**  
Pointer to the remote path/filename on the server for the file to be transferred.
- **lpath**  
Pointer to the local path/filename for the file.
- **ops**  
Pointer to options the client requests of the server. Valid options are block size and timeout, specified in the `blksize` and `timeout` fields of the `TFTP_OPTIONS` respectively. Block size must be a value between `TFTP_BLOCK_SIZE_MIN` and `TFTP_BLOCK_SIZE_MAX`. If timeout is greater the 255, `TFTP_TIMEOUT_DEFAULT` will be used per RFC 2349.
- **NU\_FAMILY\_IP**  
The family type of the remote server, either `NU_FAMILY_IP` or `NU_FAMILY_IP6`.

### Return Values

- **TFTP\_NO\_MEMORY**  
Not enough memory.
- **TFTP\_CON\_FAILURE**  
Could not establish communication with the server.
- **TFTP\_ERROR**  
General error.
- **TFTP\_FILE\_NFOUND**  
The file does not exist.
- **TFTP\_ACCESS\_VIOLATION**  
The file cannot be accessed by the user.
- **TFTP\_DISK\_FULL**  
Disk full or allocation exceeded.

- **TFTP\_BAD\_OPERATION**  
 Illegal TFTP operation.
- **TFTP\_UNKNOWN\_TID**  
 Unknown transfer ID.
- **TFTP\_FILE\_EXISTS**  
 File already exists and cannot be overwritten.
- **TFTP\_NO\_SUCH\_USER**  
 No such user.
- **TFTP\_BAD\_OPTION**  
 Illegal negotiation option.

### Example

```

INT32          bytes_sent;
NU_TFTP_OPTIONS tftp_ops;
UINT8          TFTP_Serv_IP_Addr[] = {192, 168, 1, 1};
CHAR           Demo_File_To_Test[20] = "test.txt";

/* Fill in the options. */
tftp_ops.blksize = 1024; /* Valid range (8 <= blksize <= 65464) */
tftp_ops.timeout = 60; /* Valid range (1 <= timeout <= 244) */

bytes_sent = NU_TFTPGet2(TFTP_Serv_IP_Addr, Demo_File_To_Test,
Demo_File_To_test, &tftp_ops, NU_FAMILY_IP);

```

### Related Topics

[Net API Functions](#)

[NU\\_TFTPPut2 \(IPv4/IPv6\)](#)

## NU\_TFTP\_PC\_Put2 (IPv4/IPv6)

Puts a file onto a remote TFTP server.

### Usage

```
INT32 NU_TFTP_PC_Put2 (const UINT8    *remote_ip,  
                      const CHAR    *rpath,  
                      CHAR          *lpath,  
                      TFTP_OPTIONS  *ops,  
                      INT16         NU_FAMILY_IP);
```

### Arguments

- **remote\_ip**  
Pointer to the remote TFTP server IP address.
- **rpath**  
Pointer to the remote path/filename for the file on the remote server.
- **lpath**  
Pointer to the local path/filename of the file to be transferred.
- **ops**  
Pointer to options the client requests of the server. Valid options are block size and timeout, specified in the `blksize` and `timeout` fields of the `TFTP_OPTIONS` respectively. Block size must be a value between `TFTP_BLOCK_SIZE_MIN` and `TFTP_BLOCK_SIZE_MAX`. If timeout is greater the 255, `TFTP_TIMEOUT_DEFAULT` will be used per RFC 2349.
- **NU\_FAMILY\_IP**  
The family type of the remote server, either `NU_FAMILY_IP` or `NU_FAMILY_IP6`.

### Return Values

- **TFTP\_NO\_MEMORY**  
Not enough memory.
- **TFTP\_CON\_FAILURE**  
Could not establish communication with the server.
- **TFTP\_ERROR**  
General error.
- **TFTP\_FILE\_NFOUND**  
The file does not exist.
- **TFTP\_ACCESS\_VIOLATION**  
The file cannot be accessed by the user.
- **TFTP\_DISK\_FULL**  
Disk full or allocation exceeded.

- **TFTP\_BAD\_OPERATION**  
 Illegal TFTP operation.
- **TFTP\_UNKNOWN\_TID**  
 Unknown transfer ID.
- **TFTP\_FILE\_EXISTS**  
 File already exists and cannot be overwritten.
- **TFTP\_NO\_SUCH\_USER**  
 No such user.
- **TFTP\_BAD\_OPTION**  
 Illegal negotiation option.

### Example

```

INT32          bytes_sent;
NU_TFTP_OPTIONS tftp_ops;
UINT8          TFTP_Serv_IP_Addr[] = {192, 168, 1, 1};
CHAR           Demo_File_To_Test[20] = "test.txt";

/* Fill in the options. */
tftp_ops.blksize = 1024; /* Valid range (8 <= blksize <= 65464) */
tftp_ops.timeout = 60;  /* Valid range (1 <= timeout <= 244) */

bytes_sent = NU_TFTP_C_Put2(TFTP_Serv_IP_Addr, Demo_File_To_Test,
Demo_File_To_test, &tftp_ops, NU_FAMILY_IP);

```

### Related Topics

[Net API Functions](#)

[NU\\_TFTP\\_C\\_Get2 \(IPv4/IPv6\)](#)

## NU\_Test\_Link\_Up

This function returns TRUE if the link associated with the device session handle is up.

### Usage

```
STATUS NU_Test_Link_Up (CHAR    *dev_name,  
                        UINT32  *state);
```

### Arguments

- dev\_name  
Pointer to the Ethernet device name.
- state  
Pointer to the link state: 1 = up, 0 = down.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- -1  
Otherwise.

### Example

```
UINT32      state;  
  
/* Initialize an interface. */  
. . .  
  
/* Disable the interface. */  
. . .  
  
/* Get the state of the link. */  
if (NU_Test_Link_Up("eth0") == NU_SUCCESS)  
{  
    /* Ensure the link is down. */  
    if (state == 1)  
    {  
        . . .  
    }  
}
```

### Related Topics

[Net API Functions](#)

[NU\\_Ethernet\\_Link\\_Down](#)

[NU\\_Ethernet\\_Link\\_Up](#)



## NU\_Update\_DNS\_Record

This routine is used to create, delete or update a DNS record in the local database.

### Usage

```
STATUS NU_Update_DNS_Record(UNSIGNED *handle,  
                           NU_DNS_HOST *dns_record,  
                           UINT8      action);
```

### Arguments

- handle

Pointer to the handle of the record to update. If the record is being added, this value will be filled in by the routine.

- dns\_record

Pointer to the update to make to the record:

dns\_name

Name of the record.

dns\_type

DNS\_TYPE\_A

DNS\_TYPE\_AAAA

DNS\_TYPE\_PTR

DNS\_TYPE\_SRV

DNS\_TYPE\_TXT

dns\_data

Data associated with the record. If the record is A or AAAA, this is a list of IP addresses. Otherwise, it is data specific to the record.

dns\_ttl

TTL of the record in seconds.

dns\_family

Family of the IP addresses or NU\_FAMILY\_UNSPEC if the record does not reference an IP address.

dns\_data\_len

If the record is A or AAAA, this is the number of IP addresses. Otherwise, it is the length of data in bytes.

dns\_prio

If the type is DNS\_TYPE\_SRV, the priority level of this service.

dns\_weight

If the type is DNS\_TYPE\_SRV, the weight value of this service.

dns\_port

If the type is DNS\_TYPE\_SRV, the port number over which this service can be reached.

dns\_flags

DNS\_PERMANENT\_ENTRY - the entry will not be deleted.

DNS\_UNIQUE\_RECORD - the record is unique - mDNS probing and announcing will be performed if mDNS is included in the build.

- action

Pointer to add, delete or modify the record:

NU\_DNS\_ADD

NU\_DNS\_DELETE

NU\_DNS\_UPDATE

## Return Values

- NU\_SUCCESS

The function completed successfully.

- NU\_INVALID\_PARM

One of the input parameters is invalid.

- NU\_NO\_ACTION

The caller is requesting either to add a record and a matching record that already exists, to delete a record that does not exist, or to update a record that is learned dynamically from the network or that does not exist.

- Operating system specific error code, otherwise.

## Description

When mDNS is enabled, questions for records added via this routine will be answered by the local node. The application can invoke probing and announcing on the record by setting the flag parameter to DNS\_UNIQUE\_RECORD as described here.

The TTL of any record type can be updated by this routine. Otherwise, only the data portion of the SRV and TXT records can be updated.

## Example

```
UINT8          ipv4_addr[] = {192, 168, 1, 189};
NU_DNS_HOST    new_record;
UNSIGNED       handle = 0;

new_record->dns_ttl = 120; /* 120 seconds. */
new_record->dns_type = DNS_TYPE_A;
new_record->dns_data_len = 1; /* 1 IP address in the record. */
new_record->dns_family = NU_FAMILY_IP;
new_record->dns_name = "my_record.local";
```

```
/* Perform mDNS probing and announcing. */
new_record->dns_flags = DNS_AUTHORITATIVE_RECORD;

new_record->dns_data = (CHAR*)ipv4_addr;

/* Create an A record. */
status = NU_Update_DNS_Record(&handle, &new_record, NU_DNS_ADD);

/* Update the TTL of the record. */
new_record->dns_ttl = 240; /* 240 seconds. */
status = NU_Update_DNS_Record(&handle, &new_record, NU_DNS_UPDATE);

/* Delete the record. */
status = NU_Update_DNS_Record(&handle, NU_NULL, NU_DNS_DELETE);
```

## Related Topics

[Net API Functions](#)

## NU\_Update\_Route (IPv4/IPv6)

This function updates parameters of a route as specified by the caller.

### Usage

```
STATUS NU_Update_Route (UINT8          *ip_address,  
                        UINT8          *gateway,  
                        UPDATED_ROUTE_NODE *new_route,  
                        INT16          family);
```

### Arguments

- `ip_address`  
Destination address of the route.
- `gateway`  
Pointer to gateway (next-hop) of the route.
- `new_route`  
Pointer to the data structure containing the parameters to modify in the route. The [UPDATED\\_ROUTE\\_NODE](#) data structure is defined in the [Net Data Structures](#) section.
- `family`  
Family of the destination address. Either `NU_FAMILY_IP6` for an IPv6 route or `NU_FAMILY_IP` for an IPv4 route.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID`  
The route is not valid.
- `NU_NO_MEMORY`  
Insufficient memory.

### Description

The caller can change the following parameters of a route; destination address, gateway address, submask or prefix length, interface, flags, metric, age, route tag, and path MTU.

### Example

```
UINT8 destination_ip[] = {192, 200, 100, 1};  
UINT8 subnet_mask[] = {255, 255, 255, 255};  
UINT8 gateway_ip[] = {192, 200, 100, 25};  
UPDATED_ROUTE_NODE route_info;  
STATUS status;  
  
/* Add a route to an on-link IPv4 router. */  
NU_Add_Route(destination_ip, subnet_mask, gateway_ip);
```

```
/* Disable Path MTU Discovery for the IPv4 route. */
memset(&route_info, -1, sizeof(UPDATED_ROUTE_NODE));

route_info.urt_flags = RT_STOP_PMTU;

status = NU_Update_Route(destination_ip, gateway_ip, &route_info,
                        NU_FAMILY_IP);
```

## Related Topics

[Net API Functions](#)

[NU\\_Add\\_Route6 \(IPv6\)](#)

[NU\\_Delete\\_Route \(IPv4\)](#)

[NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)

[NU\\_Add\\_Route \(IPv4\)](#)

## NU\_ZC\_Allocate\_Buffer (IPv4/IPv6)

This function provides the caller with a chain of NET buffers of length equal to the size provided to the function.

### Usage

```
INT32 NU_ZC_Allocate_Buffer (NET_BUFFER **zc_buf,  
                             UINT16    size,  
                             INT        socketd);
```

### Arguments

- **zc\_buf**  
Pointer to a pointer to the chain of NET buffers that will be returned by the function.
- **size**  
The desired length of the buffer chain to allocate.
- **socketd**  
The socket descriptor for the communication.

### Return Values

- The length of the buffer chain allocated and the pointer to the buffer chain passed in will point to the newly allocated chain of buffers.  
On successful completion. Note that the length of the buffer chain may not be equal to size if the socket is a TCP socket and the TCP MSS is less than the size input parameter. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes.
- **NU\_INVALID\_SOCKET**  
The socket descriptor provided does not match a socket descriptor in the system.
- **NU\_INVALID\_PARM**  
If the socket descriptor is a TCP socket the connection is not established, or the size input parameter is zero.
- **NU\_NO\_BUFFERS**  
There are no buffers available in the system and the socket is a non-blocking socket.
- **NU\_SOCKET\_CLOSED**  
The socket was closed by another task while this task was suspended waiting for buffers.

### Description

If the socket is a TCP socket, a connection must already be established on the socket, and the maximum size buffer chain to be allocated by the routine is the negotiated TCP MSS of the connection regardless of the user's size input parameter.

## Example

```

VOID TCP_Client_Task(UINT16 nbytes, struct addr_struct &servaddr)
{
    INT                tcp_client_socket;
    INT32              bytes_to_send;
    NET_BUFFER         *tcp_tx_buffer;

    /* Open a connection via the socket interface. */
    tcp_client_socket = NU_Socket(NU_FAMILY_IP,
                                  NU_TYPE_STREAM, 0);

    if (tcp_client_socket >= 0)
    {
        /* Establish a connection with the server. */
        if (NU_Connect(tcp_client_socket, servaddr, 0) >= 0)
        {
            /* Enable Zero Copy mode on the socket. */
            NU_Fcntl(tcp_client_socket, NU_SET_ZC_MODE,
                     NU_ZC_ENABLE);

            /* Get a buffer for the data and the total number
             * of bytes that will fit in this buffer. The
             * requested size of the buffer is nbytes.
             */
            bytes_to_send =
                NU_ZC_Allocate_Buffer(&tcp_tx_buffer,
                                       nbytes, tcp_client_socket);

            .
            .
            .

            } /* if (NU_Connect) */
        } /* if (tcp_client_socket >= 0) */
    } /* TCP_Client_Task */
}

```

## Related Topics

### Net API Functions

<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>
<a href="#">NU_ZC_Send (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>
<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>	

## NU\_ZC\_BUF\_LEN (IPv4/IPv6)

This function returns the total number of bytes of data in the provided buffer chain.

### Usage

```
UINT32 NU_ZC_BUF_LEN (NET_BUFFER *zc_buf);
```

### Arguments

- `zc_buf`  
Pointer to a chain of Nucleus NET buffers. The function returns the total number of bytes in the buffer chain.

### Return Values

- Number of bytes
- 0  
If an error occurred.

### Description

This routine must only be used with Zero Copy buffers that have been received from the stack via a call to [NU\\_ZC\\_Recv \(IPv4/IPv6\)](#) or [NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#). This function must not be used with Zero Copy buffers that are being used for transmission.

### Example

```
UINT16 bytes_received;  
NET_BUFFER *zc_buf;  
  
/* Receive data from a client. */  
.  
.  
.  
  
/* Determine how many bytes were received. */  
bytes_received = NU_ZC_BUF_LEN(zc_buf);
```

### Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>
<a href="#">NU_ZC_Send (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>



## NU\_ZC\_Deallocate\_Buffer (IPv4/IPv6)

This function returns the buffer passed in to the appropriate buffer list, after which, the buffer will be available to be used by other processes in the system. This function must be called by the application after receiving data in Zero Copy mode and after unsuccessful transmission of data in Zero Copy mode.

### Usage

```
STATUS NU_ZC_Deallocate_Buffer (NET_BUFFER *zc_buf);
```

### Arguments

- `zc_buf`  
Pointer to the chain of NET buffers to free.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `NU_INVALID_PARM`  
The buffer pointer passed in is NULL.

### Example

```
VOID TCP_Server_Task(UINT16 nbytes)
{
    INT          tcp_client_socket;
    NET_BUFFER    *tcp_rx_buffer;
    UINT16        bytes_recv;

    /* Create socket, set socket as Zero Copy, accept incoming
     * connection.
     */

    bytes_recv = NU_ZC_Recv(tcp_client_socket, &tcp_rx_buffer,  nbytes, 0);

    /* The application is responsible for freeing the
     * Zero Copy buffer upon successful receipt of data.
     */
    if (bytes_recv >= 0)
    {
        /* Process the data. */

        /* Free the NET Buffer. */
        NU_ZC_Deallocate_Buffer(tcp_rx_buffer);

    } /* if (bytes_recv >= 0)
    /* TCP_Server_Task */
```

## Related Topics

[Net API Functions](#)

[NU\\_ZC\\_Recv \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_DATA \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Allocate\\_Buffer \(IPv4/IPv6\)](#)

[NU\\_ZC\\_BUF\\_LEN \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_NEXT \(IPv4/IPv6\)](#)


[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## NU\_ZC\_Recv (IPv4/IPv6)

This function is responsible for receiving data across a network during a connection-oriented transfer using Zero Copy Buffers. Both the client and the server may call it.

---

**Note**

 The application is responsible for deallocating the NET Buffer chain returned from this function via a call to `NU_ZC_Deallocate_Buffer`.

---

### Usage

```
INT32 NU_ZC_Recv (INT          socketd,
                  NET_BUFFER **zc_buf,
                  UINT16       nbytes,
                  INT16        flags);
```

### Arguments

- **socketd**  
 Specifies a socket descriptor.
- **zc\_buf**  
 Pointer to a pointer to the NET buffer.
- **nbytes**  
 This parameter is unused for Zero Copy sockets since the routine will return all data that has been received on the socket.
- **flags**  
 Used for socket compatibility, but it is not currently in use

### Return Values

- The number of bytes transferred.  
 Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- **NU\_NOT\_CONNECTED**  
 The connection is broken for some reason. Stop using the socket; it is best to close it.
- **NU\_INVALID\_SOCKET**  
 The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- **NU\_NO\_PORT\_NUMBER**  
 No local port number was stored in the socket descriptor.
- **NU\_WOULD\_BLOCK**  
 Socket is non-blocking, but blocking is required to complete the requested action.

- NU\_NO\_ROUTE\_TO\_HOST  
This is an icmp\_error if no route to the host exists.
- NU\_CONNECTION\_REFUSED  
This is an icmp\_error if the connection is refused.
- NU\_MSG\_TOO\_LONG  
This is an icmp\_error if the message is too large.
- NU\_CONNECTION\_TIMED\_OUT  
TCP Keep-Alive packets found that the connection has timed out.
- NU\_DEST\_UNREACH\_ADMIN  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_ADDRESS  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_PORT  
An ICMP Error Code was received on the socket.
- NU\_TIME\_EXCEED\_HOPLIMIT  
An ICMP Error Code was received on the socket.
- NU\_TIME\_EXCEED\_REASM  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_HEADER  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_NEXT\_HDR  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_OPTION  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_NET  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_HOST  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_PROT  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_FRAG  
An ICMP Error Code was received on the socket.

- **NU\_DEST\_UNREACH\_SRCFAIL**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB**  
An ICMP Error Code was received on the socket.
- **NU\_SOURCE\_QUENCH**  
An ICMP Error Code was received on the socket.

## Example

```
VOID TCP_Server_Task(struct addr_struct &servaddr)
{
    INT          socketd;
    INT          bytes_received;
    NET_BUFFER    *tcp_rx_buffer;

    /* Open a connection via the socket interface. */
    socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

    if (socketd >= 0)
    {
        /* Enable Zero Copy mode on the socket. */
        NU_Fcntl(socketd, NU_SET_ZC_MODE, NU_ZC_ENABLE);

        /* Bind our address to the socket. */
        NU_Bind(socketd, servaddr, 0);

        for (;;)
        {
            /* Get a string from the client. */
            bytes_received =
                NU_ZC_Recv(socketd, &tcp_rx_buffer, 0, 0);

            if (bytes_received >= 0)
            {
                /* Process the data. */

                /* Free the buffer. */
                NU_ZC_Deallocate_Buffer(tcp_rx_buffer);

                } /* if (bytes_received >= 0) */
            } /* for (;;) */
        } /* if (socketd >= 0) */
    } /* TCP_Server_Task */
}
```

## Related Topics

[Net API Functions](#)

[NU\\_ZC\\_Allocate\\_Buffer \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_DATA \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send \(IPv4/IPv6\)](#)

[NU\\_ZC\\_BUF\\_LEN \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_NEXT \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## NU\_ZC\_Recv\_From (IPv4/IPv6)

This function is responsible for receiving data across a network during a connectionless transfer using Zero Copy Buffers. Both the client and server may call it.

### Note



The application is responsible for deallocating the NET Buffer chain returned from this function via a call to `NU_ZC_Deallocate_Buffer`.

## Usage

```
INT32 NU_ZC_Recv_From (INT          socketd,  
                      NET_BUFFER   **zc_buf,  
                      UINT16        nbytes,  
                      INT16         flags,  
                      struct addr_struct *from,  
                      INT16         addrlen);
```

## Arguments

- `socketd`  
Specifies a socket descriptor.
- `zc_buf`  
Pointer to a pointer to the NET buffer.
- `nbytes`  
This parameter is unused for Zero Copy sockets since the routine will return all data that has been received on the socket.
- `flags`  
Used for socket compatibility. This is not currently being used.
- `from`  
Pointer to the source protocol-specific [addr\\_struct](#) structure.
- `addrlen`  
This parameter is reserved for future use. A value of zero should be used.

## Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.

- NU\_NO\_PORT\_NUMBER  
No local port number was stored in the socket descriptor.
- NU\_INVALID\_PARM  
The [addr\\_struct](#) “from” is NULL.
- NU\_NOT\_CONNECTED  
The socket is not connected.
- NU\_WOULD\_BLOCK  
Socket is non-blocking, but blocking is required to complete the requested action.
- NU\_NO\_DATA\_TRANSFER  
The data transfer was not completed.
- NU\_DEVICE\_DOWN  
The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.
- NU\_DEST\_UNREACH\_ADMIN  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_ADDRESS  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_PORT  
An ICMP Error Code was received on the socket.
- NU\_TIME\_EXCEED\_HOPLIMIT  
An ICMP Error Code was received on the socket.
- NU\_TIME\_EXCEED\_REASM  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_HEADER  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_NEXT\_HDR  
An ICMP Error Code was received on the socket.
- NU\_PARM\_PROB\_OPTION  
An ICMP Error Code was received on the socket.
- NU\_DEST\_UNREACH\_NET  
An ICMP Error Code was received on the socket.



- **NU\_DEST\_UNREACH\_HOST**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_PROT**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_FRAG**  
An ICMP Error Code was received on the socket.
- **NU\_DEST\_UNREACH\_SRCFAIL**  
An ICMP Error Code was received on the socket.
- **NU\_PARM\_PROB**  
An ICMP Error Code was received on the socket.
- **NU\_SOURCE\_QUENCH**  
An ICMP Error Code was received on the socket.

### Example

```

VOID UDP_Server_Task(struct addr_struct &servaddr)
{
    INT                socketd;
    struct addr_struct  cliaddr;
    INT                bytes_received;
    INT16              clilen;
    NET_BUFFER          *udp_rx_buffer;

    /* open a connection via the socket interface. */
    socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_DGRAM, 0);

    if (socketd >= 0)
    {
        /* Enable Zero Copy mode on the socket. */
        NU_Fcntl(socketd, NU_SET_ZC_MODE, NU_ZC_ENABLE);

        /* Bind our address to the socket. */
        NU_Bind(socketd, servaddr, 0);

        for (;;)
        {
            /* Get a string from the client. */
            bytes_received =
                NU_ZC_Recv_From(socketd, &udp_rx_buffer, 0, 0,
                                &cliaddr, &clilen);

            if (bytes_received >= 0)
            {
                /* Process the data. */

                /* Free the buffer. */
                NU_ZC_Deallocate_Buffer(udp_rx_buffer);
            }
        }
    }
}

```

```
        } /* if (bytes_received >= 0) */
    }    /* for (;;) */
}       /* if (socketd >= 0) */
        /* UDP_Server_Task */
```

## Related Topics

[Net API Functions](#)

[NU\\_ZC\\_Allocate\\_Buffer \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_DATA \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send \(IPv4/IPv6\)](#)

[NU\\_ZC\\_BUF\\_LEN \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_NEXT \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## NU\_ZC\_SEGMENT\_BYTES\_AVAIL (IPv4/IPv6)

This function returns the total number of bytes of data in the provided segment of a buffer chain.

### Usage

```
UINT32 NU_ZC_SEGMENT_BYTES_AVAIL (NET_BUFFER *zc_buf);
```

### Arguments

- `zc_buf`  
Pointer to a segment of a buffer chain.

### Return Values

- The total number of bytes in the segment of a buffer chain.

### Description

This routine must only be used with Zero Copy buffers that have been received from the stack via a call to `NU_ZC_Recv()` or `NU_ZC_Recv_From()`. This function must not be used with Zero Copy buffers that are being used for transmission.

### Example

```
INT          file_desc;
NET_BUFFER  *seg_ptr, *zc_buffer;
CHAR        *data_ptr;
UINT16      bytes_avail;

/* Receive data from a client. */

.
.
.

/* Open a new file in which to store the data. */
file_desc = NU_Open("test.txt", PO_RDWR|PO_CREAT, PS_IWRITE);

/* Get a pointer to the buffer chain of data. */
seg_ptr = zc_buffer;

while (seg_ptr)
{
    /* Get the first segment in the buffer chain. */
    data_ptr = NU_ZC_SEGMENT_DATA(seg_ptr);

    /* Determine how many bytes are in this segment. */
    bytes_avail = NU_ZC_SEGMENT_BYTES_AVAIL(seg_ptr);

    /* Write the data to the file. */
    NU_Write(file_desc, data_ptr, bytes_avail);

    /* Get a pointer to the next segment. */
    seg_ptr = NU_ZC_SEGMENT_NEXT(seg_ptr);
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>
<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>
<a href="#">NU_ZC_Send (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>

## NU\_ZC\_SEGMENT\_BYTES\_LEFT (IPv4/IPv6)

This function returns the number of bytes that can be copied into the segment.

### Note



The application layer must completely fill each segment in the chain before moving on to the next segment in the chain. Only the last segment in the chain may be filled in only partially.

### Usage

```
UINT32 NU_ZC_SEGMENT_BYTES_LEFT (NET_BUFFER *zc_buf,  
                                NET_BUFFER *seg_ptr,  
                                INT      socketd);
```

### Arguments

- `zc_buf`  
Pointer to the head of the buffer chain of which `seg_ptr` is a segment.
- `seg_ptr`  
Pointer to a segment in the buffer chain.
- `socketd`  
The socket descriptor for the communication.

### Return Values

- Upon successful completion, this function returns the total number of bytes that can be copied into the segment of a buffer chain.
- 0  
Otherwise.

### Example

```
VOID TCP_Client_Task()  
{  
    INT          tcp_client_socket;  
    INT32        bytes_to_send;  
    NET_BUFFER   *tcp_tx_buffer, *seg_ptr;  
    CHAR         *data_ptr;  
    UINT32       bytes_left, bytes_copied;  
  
    /* Create socket, set socket as Zero Copy, connect to server, get  
     * Zero Copy buffer. */  
    .  
    .  
    .  
  
    /* If a buffer chain was successfully allocated */  
    if (bytes_to_send > 0)  
    {  
        /* Get a pointer to the new buffer chain. */  
    }
```

```
    seg_ptr = tcp_tx_buffer;

    bytes_copied = 0;

    /* Fill the buffer chain with the data, one segment at a time. */
    while (seg_ptr)
    {
        /* Get the first segment in the buffer chain */
        data_ptr = NU_ZC_SEGMENT_DATA(seg_ptr);

        /* Determine how many bytes will fit in this
         * segment.
         */
        bytes_left = NU_ZC_SEGMENT_BYTES_LEFT(tcp_tx_buffer, seg_ptr,
                                              tcp_client_socket);

        /* Determine how many bytes to copy into this segment. */
        if (bytes_left > (bytes_to_send - bytes_copied))
            bytes_left = bytes_to_send - bytes_copied;
        /* Copy the data into the segment. */
        memset(data_ptr, 'A', bytes_left);

        /* Get a pointer to the next segment. */
        seg_ptr = NU_ZC_SEGMENT_NEXT(seg_ptr);
    }
}
```

## Related Topics


<a href="#">Net API Functions</a>	<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>
<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>
<a href="#">NU_ZC_Send (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>

## NU\_ZC\_SEGMENT\_DATA (IPv4/IPv6)

This function returns a pointer to the beginning of the data section of a segment in a chain of Nucleus NET buffers.

---

**Note**

 The application layer must completely fill each segment in the chain before moving on to the next segment in the chain. Only the last segment in the chain may be filled in only partially.

---

### Usage

```
CHAR *NU_ZC_SEGMENT_DATA (NET_BUFFER *seg_ptr);
```

### Arguments

- `seg_ptr`  
Pointer to a segment in the buffer chain.

### Return Values

- The function returns a pointer to the beginning of the data section of a segment in a chain of NET buffers.

### Example

```
VOID TCP_Client_Task()
{
    INT                tcp_client_socket;
    INT32              bytes_to_send;
    NET_BUFFER          *tcp_tx_buffer, *seg_ptr;
    CHAR                *data_ptr;
    UINT16              bytes_left, bytes_copied;

    /* Create socket, set socket as Zero Copy, connect to
     * server, get Zero Copy buffer.
     */
    .
    .
    .

    /* If a buffer chain was successfully allocated */
    if (bytes_to_send > 0)
    {
        /* Get a pointer to the new buffer chain. */
        seg_ptr = tcp_tx_buffer;

        bytes_copied = 0;

        /* Fill the buffer chain with the data, one
         * segment at a time.
         */
        while (seg_ptr)
        {
            /* Get the first segment in the buffer.
```

```
        * chain
        */
data_ptr = NU_ZC_SEGMENT_DATA(seg_ptr);

/* Determine how many bytes will fit in this
 * segment.
 */
bytes_left = NU_ZC_SEGMENT_BYTES_LEFT(tcp_tx_buffer,
                                       seg_ptr,
                                       tcp_client_socket);

/* Determine how many bytes to copy into
 * this segment.
 */
if (bytes_left > (bytes_to_send - bytes_copied))
    bytes_left = bytes_to_send - bytes_copied;

/* Copy the data into the segment. */
memset(data_ptr, 'A', bytes_left);

/* Get a pointer to the next segment. */
seg_ptr = NU_ZC_SEGMENT_NEXT(seg_ptr);
    }
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>
<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>
<a href="#">NU_ZC_Send (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>



## NU\_ZC\_SEGMENT\_NEXT (IPv4/IPv6)

This function returns a pointer to the next segment in a chain of Nucleus NET buffers.

### Usage

```
NET_BUFFER *NU_ZC_SEGMENT_NEXT (NET_BUFFER *seg_ptr);
```

### Arguments

- `seg_ptr`  
Pointer to a segment in the buffer chain.

### Return Values

- A pointer to the next segment in a chain of Nucleus NET buffers.

### Example

```
VOID TCP_Client_Task()
{
    INT                tcp_client_socket;
    INT32              bytes_to_send;
    NET_BUFFER          *tcp_tx_buffer, *seg_ptr;
    CHAR                *data_ptr;
    UINT32              bytes_left, bytes_copied;

    /* Create socket, set socket as Zero Copy, connect to server, get
       Zero Copy buffer. */
    .
    .
    .

    /* If a buffer chain was successfully allocated */
    if (bytes_to_send > 0)
    {
        /* Get a pointer to the new buffer chain. */
        seg_ptr = tcp_tx_buffer;

        bytes_copied = 0;

        /* Fill the buffer chain with the data, one segment at a time. */
        while (seg_ptr)
        {
            /* Get the first segment in the buffer chain. */
            data_ptr = NU_ZC_SEGMENT_DATA(seg_ptr);

            /* Determine how many bytes will fit in this segment. */

            bytes_left = NU_ZC_SEGMENT_BYTES_LEFT(tcp_tx_buffer, seg_ptr,
                                                    tcp_client_socket);

            /* Determine how many bytes to copy into this segment. */
            if (bytes_left > (bytes_to_send - bytes_copied))
                bytes_left = bytes_to_send - bytes_copied;

            /* Copy the data into the segment. */
```

```
        memset(data_ptr, 'A', bytes_left);

        /* Get a pointer to the next segment. */
        seg_ptr = NU_ZC_SEGMENT_NEXT(seg_ptr);
    }
}
```

## Related Topics

[Net API Functions](#)

[NU\\_ZC\\_Allocate\\_Buffer \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv\\_From \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send \(IPv4/IPv6\)](#)

[NU\\_ZC\\_BUF\\_LEN \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Recv \(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL  
\(IPv4/IPv6\)](#)

[NU\\_ZC\\_SEGMENT\\_DATA \(IPv4/IPv6\)](#)

[NU\\_ZC\\_Send\\_To \(IPv4/IPv6\)](#)

## NU\_ZC\_Send (IPv4/IPv6)

This function is responsible for transmitting data across a network during a connection-oriented transfer using Zero Copy Buffers. Both a client and server may use the function.

### Note



If this function returns a negative value, the application is responsible for deallocating the Nucleus NET buffer chain via a call to `NU_ZC_Deallocate_Buffer`. However, if this function returns a positive value that does not match the number of bytes passed in to be transmitted (`nbytes`), the function already deallocated the entire chain of buffers. The application must not attempt to deallocate the buffers again.

## Usage

```
INT32 NU_ZC_Send (INT          socketd,  
                  NET_BUFFER *zc_buf,  
                  UINT16      nbytes,  
                  INT16       flags);
```

## Arguments

- `socketd`  
Specifies a socket descriptor.
- `zc_buf`  
Pointer to the NET buffer.
- `nbytes`  
Specifies the number of bytes of data. Valid values for this parameter are in the range 0-`IP_MAX_DATA_SIZE` (65,495).
- `flags`  
This parameter is used for socket compatibility.

## Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- `NU_INVALID_SOCKET`  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- `NU_NO_PORT_NUMBER`  
No local port number was stored in the socket descriptor.

- **NU\_NOT\_CONNECTED**  
The data transfer was not completed. This probably occurred because the connection was closed for some reason.
- **NU\_NO\_ROUTE\_TO\_HOST**  
This is an icmp\_error if no route to the host exists.
- **NU\_CONNECTION\_REFUSED**  
This is an icmp\_error if the connection is refused.
- **NU\_MSG\_TOO\_LONG**  
This is an icmp\_error if the message is too large.
- **NU\_WOULD\_BLOCK**  
Socket is non-blocking, but blocking is required to complete the requested action.

### Example

```
VOID TCP_Client_Task(UINT16 nbytes, struct addr_struct &servaddr)
{
    INT          tcp_client_socket;
    INT32        bytes_sent, bytes_to_send;
    NET_BUFFER   *tcp_tx_buffer;

    /* Open a connection via the socket interface. */
    tcp_client_socket = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);

    if (tcp_client_socket >= 0)
    {
        /* Establish a connection with the server. */
        if (NU_Connect(tcp_client_socket, servaddr, 0) >= 0)
        {
            /* Enable Zero Copy mode on the socket. */
            NU_Fcntl(tcp_client_socket, NU_SET_ZC_MODE,
                    NU_ZC_ENABLE);

            /* Get a Zero Copy buffer and fill it with data. */
            .
            .
            .

            /* Send the datagram. */
            bytes_sent =
            NU_ZC_Send(tcp_client_socket, tcp_tx_buffer,
                      (UINT16)bytes_to_send, 0);

            /* If the data could not be sent, the application is
             * responsible for freeing the buffer. */
            if (bytes_sent < 0) NU_ZC_Deallocate_Buffer(tcp_tx_buffer);

        } /* if (NU_Connect()) */
    } /* if (tcp_client_socket) */
} /* TCP_Client_Task */
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>
<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send_To (IPv4/IPv6)</a>

## NU\_ZC\_Send\_To (IPv4/IPv6)

This function is used by UDP applications to transmit data across a network during a connectionless transfer using Zero Copy Buffers.

### Usage

```
INT32 NU_ZC_Send_To (INT          socketd,  
                    NET_BUFFER  *zc_buf,  
                    UINT16      nbytes,  
                    INT16       flags,  
                    struct addr_struct *to,  
                    INT16       addrlen);
```

### Arguments

- **socketd**  
Specifies a socket descriptor.
- **zc\_buf**  
Pointer to the NET buffer.
- **nbytes**  
Specifies the number of bytes of data.
- **flags**  
This parameter is used for socket compatibility.
- **to**  
Pointer to the destination's protocol-specific [addr\\_struct](#) structure.
- **addrlen**  
This parameter is reserved for future use. A value of zero should be used.

### Return Values

- The number of bytes transferred.  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code or one of the following Nucleus NET error codes:
- **NU\_INVALID\_SOCKET**  
The socket parameter was not a valid socket value, or it had not been previously allocated via the `NU_Socket` call.
- **NU\_INVALID\_PARM**  
The [addr\\_struct](#) to is `NU_NULL`.
- **NU\_NOT\_CONNECTED**  
If the socket is not connected.


- **NU\_NO\_PORT\_NUMBER**  
 No local port number was stored in the socket descriptor.
- **NU\_NO\_DATA\_TRANSFER**  
 The data transfer was not completed.
- **NU\_INVALID\_ADDRESS**  
 The address passed in was most likely incomplete (for example, missing the IP or port number).
- **NU\_WOULD\_BLOCK**  
 Socket is non-blocking, but blocking is required to complete the requested action.
- **NU\_DEVICE\_DOWN**  
 The device that this socket was communicating over has gone down. If the device is a PPP device it is likely the physical connection has been broken. If the device is Ethernet and DHCP is being used, the lease of the IP address may have expired.

## Description

Both UDP clients and UDP servers may call `NU_Send_To`. The largest block of data that can be transmitted is 65,507 bytes. There are 28 bytes of overhead for the IP and UDP headers.

---

**Note**

 If this function returns a negative value, the application is responsible for deallocating the Nucleus NET buffer chain via a call to `NU_ZC_Deallocate_Buffer`. However, if this function returns a positive value that does not match the number of bytes passed in to be transmitted (`nbytes`), the function already deallocated the entire chain of buffers. The application must not attempt to deallocate the buffers again.

---

## Example

```
VOID UDP_Client_Task(UINT16 nbytes, struct addr_struct &servaddr)
{
    INT                udp_client_socket;
    INT32              bytes_sent, bytes_to_send;
    NET_BUFFER         *udp_tx_buffer;

    /* Create a socket for the transmission. */
    udp_client_socket = NU_Socket(NU_FAMILY_IP,
                                NU_TYPE_DGRAM, 0);


    if (udp_client_socket >= 0)
    {
        /* Enable Zero Copy mode on the socket. */
        NU_Fcntl(udp_client_socket, NU_SET_ZC_MODE,
                NU_ZC_ENABLE);

        /* Get a Zero Copy buffer and fill it with data. */
        .
        .
    }
}
```

```
.  
  
/* Send the datagram. */  
bytes_sent =  
NU_ZC_Send_To(udp_client_socket, udp_tx_buffer,  
               (UINT16)bytes_to_send, 0, &servaddr, 0);  
  
/* If the data could not be sent, the application is  
   responsible for freeing the buffer. */  
if (bytes_sent < 0) NU_ZC_Deallocate_Buffer(udp_tx_buffer);  
}  
}
```

## Related Topics

<a href="#">Net API Functions</a>	<a href="#">NU_ZC_BUF_LEN (IPv4/IPv6)</a>
<a href="#">NU_ZC_Allocate_Buffer (IPv4/IPv6)</a>	<a href="#">NU_ZC_Recv (IPv4/IPv6)</a>
<a href="#">NU_ZC_Recv_From (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_BYTES_AVAIL (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_BYTES_LEFT (IPv4/IPv6)</a>	<a href="#">NU_ZC_SEGMENT_DATA (IPv4/IPv6)</a>
<a href="#">NU_ZC_SEGMENT_NEXT (IPv4/IPv6)</a>	<a href="#">NU_ZC_Send (IPv4/IPv6)</a>

 Using this implementation to send data from multiple threads over the same socket, may cause undefined behavior. This is because the NET buffer currently holds the socket pointer but does not keep track of which thread is associated with that buffer.

## NET User Management

The User Management (UM) subsystem provides a centralized set of functions to manage Nucleus NET user accounts. The UM subsystem is a simple database with user name, password and permission fields. It has been set up to be used by Nucleus PPP, Nucleus FTP, and Nucleus WebServ, but additional modules can be added by modifying *um\_defs.h*.

The following services are supported by the UM subsystem and are described through API function calls within this chapter:

- Add a new user to the UM database.
- Delete a user from the UM database.
- Delete a user's permissions from the UM database.
- Get the user record for any user in the UM database.



- Get the first user record from the UM database.
- Get the next user record from the UM database.
- Get the number of users with the specified permissions.
- Validate the password and permissions for a given user.

## User Management API Functions

This section describes the APIs in the UM database.

- [UM\\_Add\\_User](#)
- [UM\\_Del\\_User](#)
- [UM\\_Delete\\_User\\_Permissions](#)
- [UM\\_Find\\_User](#)
- [UM\\_Find\\_User\\_First](#)
- [UM\\_Find\\_User\\_Next](#)
- [UM\\_Get\\_User\\_Count](#)
- [UM\\_Validate\\_User](#)

## UM\_Add\_User

This routine adds a new user account or updates an existing account in the UM database.

### Usage

```
STATUS UM_Add_User (const CHAR *name,  
                   const CHAR *pw,  
                   UINT32     pv,  
                   UINT16     flags);
```

### Arguments

- name  
Pointer to an ASCII character string representing the user account.
- pw  
Pointer to an ASCII character string representing the password.
- pv  
User permissions:
  - UM\_FTP – give user access to FTP service.
  - UM\_PPP – give user access to PPP service.
  - UM\_WEB – give user access to WebServer service.
  - UM\_ALL – give user access to ALL services.
- flags  
Flags indicating what action to take:
  - UM\_ADD\_MODE - Adds a new user account.
  - UM\_UPDATE\_MODE - Updates an existing user account.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- NU\_NO\_MEMORY  
Memory allocation failure on a new user data record.
- UM\_INVALID\_NAME  
The user name is invalid. Note that UM\_Check\_User\_Name performs only string length checking. You should add any additional validation tests to ensure user names are legal.
- UM\_INVALID\_PASSWORD  
The password is invalid. Note that the UM\_Check\_Password function does only string length checking. You should add any additional validation tests to ensure passwords are legal.

- **UM\_USER\_EXISTS**  
User name already exists in the database, and mode equals UM\_ADD\_MODE.
- **UM\_USER\_UNKNOWN**  
User name is not in UM database and mode equals UM\_UPDATE\_MODE

### Example

```
STATUS    status;  
status = UM_Add_User ("John", "doe", UM_FTP, UM_ADD_MODE);
```

### Related Topics

[NET User Management](#)

[UM\\_Delete\\_User\\_Permissions](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Del\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Del\_User

This routine deletes an existing user account from the UM database.

### Usage

```
STATUS UM_Del_User (const CHAR *name);
```

### Arguments

- **name**  
Pointer to an ASCII character string representing the user account.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **UM\_USER\_UNKNOWN**  
User account does not exist in the database.

### Example

```
STATUS    status;  
status = UM_Del_User ("John");
```

### Related Topics

[NET User Management](#)

[UM\\_Delete\\_User\\_Permissions](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Add\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Delete\_User\_Permissions

This routine deletes permissions from an existing user account in the UM database. If all permissions are deleted, the user's account is also deleted.

### Usage

```
STATUS UM_Delete_User_Permissions (const CHAR *name,  
                                  UINT32      permissions);
```

### Arguments

- **name**  
Pointer to an ASCII character string representing the user account.
- **permissions**

The permission(s) to delete from the user:

UM\_FTP – remove access from FTP service

UM\_PPP – remove access from PPP service

UM\_WEB – remove access from WebServer service

UM\_ALL - remove access from ALL services

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **UM\_USER\_UNKNOWN**  
User account does not exist in the database.

### Example

```
STATUS      status;  
  
/* Deny "John" access to FTP and PPP. */  
status = UM_Del_User ("John", UM_FTP | UM_PPP);
```

### Related Topics

[NET User Management](#)

[UM\\_Del\\_User](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Add\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Find\_User

This routine retrieves the UM database record for the specified user.

### Usage

```
STATUS UM_Find_User (const CHAR *name,  
                    UM_USER    *dp);
```

### Arguments

- name  
Pointer to an ASCII character string representing the user account.
- dp  
Pointer to [UM\\_USER](#) structure defined in the [Net Data Structures](#) section.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- UM\_USER\_UNKNOWN  
User account was not found in the database.

### Example

```
STATUS      status;  
UM_USER_TYPE *entry = (UM_USER_TYPE *) NU_NULL;  
  
/* Get the first entry from the UM database. */  
status = UM_Find_User ("John", &entry);
```

### Related Topics

[NET User Management](#)

[UM\\_Del\\_User](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Add\\_User](#)

[UM\\_Delete\\_User\\_Permissions](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Find\_User\_First

This routine retrieves the first user account from the UM database.

### Usage

```
STATUS UM_Find_User_First (UM_USER *dp,  
                          UINT32  service);
```

### Arguments

- `dp`  
Pointer to the [UM\\_USER](#) structure, defined in the [Net Data Structures](#) section.
- `service`  
The service associated with the first user to find. Services are defined in the file *include/networking/um\_defs.h*. By default, Nucleus NET currently supports UM\_FTP, UM\_PPP, UM\_WEB, and UM\_ALL. You can add custom-defined services to this file as well.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `UM_USER_UNKNOWN`  
No user accounts have been added to the UM database.

### Example

```
STATUS      status;  
UM_USER_TYPE *entry = (UM_USER_TYPE *) NU_NULL;  
  
/* Get the first entry from the UM database that is registered  
   for the FTP service. */  
status = UM_Find_User_First (&entry, UM_FTP);
```

### Related Topics

[NET User Management](#)

[UM\\_Del\\_User](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Delete\\_User\\_Permissions](#)

[UM\\_Add\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Find\_User\_Next

This routine retrieves the next user account after the current user account in the UM database.

### Usage

```
STATUS UM_Find_User_Next (const CHAR *name,  
                          UM_USER    *dp,  
                          UINT32     service);
```

### Arguments

- name  
Pointer to an ASCII character string representing the user account.
- dp  
Points to the [UM\\_USER](#) structure defined in the [Net Data Structures](#) section.
- service  
The service associated with the next user to find. Services are defined in the file *include/networking/um\_defs.h*. By default, Nucleus NET currently supports UM\_FTP, UM\_PPP, UM\_WEB, and UM\_ALL. You can add custom-defined services to this file as well.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- UM\_USER\_UNKNOWN  
User account is not defined in the UM database.

### Example

```
STATUS      status;  
UM_USER_TYPE *entry = (UM_USER_TYPE *) NU_NULL;  
  
/* Get the entry next to "John" from the UM database that is  
   registered for the FTP service.*/  
status = UM_Find_User_Next ("John", &entry, UM_FTP);
```

### Related Topics

[NET User Management](#)

[UM\\_Del\\_User](#)

[UM\\_Get\\_User\\_Count](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Add\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Delete\\_User\\_Permissions](#)



## UM\_Get\_User\_Count

This routine returns the number of user accounts with the specified permissions.

### Usage

```
UINT16 UM_Get_User_Count (UINT32 pv);
```

### Arguments

- pv  
Permission value:  
UM\_FTP, UM\_PPP, UM\_WEB, or UM\_ALL or use bitwise OR to get any combination of UM\_FTP, UM\_PPP, and UM\_WEB.

### Return Values

- The total number of user accounts in the database with the specified permissions, upon successful completion.

### Example

```
UINT16 count;  
  
/* Count number of NET user accounts with UM_PPP permission. */  
count = UM_Get_User_Count (UM_PPP);
```

### Related Topics

[NET User Management](#)

[UM\\_Del\\_User](#)

[UM\\_Delete\\_User\\_Permissions](#)

[UM\\_Find\\_User\\_First](#)

[UM\\_Add\\_User](#)

[UM\\_Find\\_User](#)

[UM\\_Validate\\_User](#)

[UM\\_Find\\_User\\_Next](#)

## UM\_Validate\_User

This routine validates the password and permission fields for the specified user account.

### Usage

```
STATUS UM_Validate_User (const CHAR *name,  
                        const CHAR *pw,  
                        UINT32      pv);
```

### Arguments

- name  
Pointer to an ASCII character string representing the user account.
- pw  
Pointer to an ASCII character string representing the password.
- pv  
Permissions:
  - UM\_FTP - check user access to FTP service
  - UM\_PPP - check user access to PPP service
  - UM\_WEB - check user access to WebServer service
  - UM\_ALL - check user access to ALL NET services

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- UM\_PW\_MISMATCH  
The password supplied does not match the password stored in the UM database for the specified user account.
- UM\_PV\_MISMATCH  
The permission supplied does not match the permission stored in the UM database for the specified user account.
- UM\_USER\_UNKNOWN  
Supplied user account is not defined in the UM database.

### Example

```
STATUS status;  
  
status = UM_Validate_User ("John", "doe", UM_FTP);
```

## Related Topics

<a href="#">NET User Management</a>	<a href="#">UM_Add_User</a>
<a href="#">UM_Del_User</a>	<a href="#">UM_Find_User</a>
<a href="#">UM_Get_User_Count</a>	<a href="#">UM_Delete_User_Permissions</a>
<a href="#">UM_Find_User_First</a>	<a href="#">UM_Find_User_Next</a>

## NET APIs Available to a Nucleus Process.

The following is a list of the NET APIs available from a Nucleus Process:

- [NETBOOT\\_Wait\\_For\\_Network\\_Up](#)
- [NU\\_Abort \(IPv4/IPv6\)](#)
- [NU\\_Accept \(IPv4/IPv6\)](#)
- [NU\\_Add\\_DNS\\_Server2 \(IPv4/IPv6\)](#)
- [NU\\_Add\\_Route \(IPv4\)](#)
- [NU\\_ARP\\_Update](#)
- [NU\\_Attach\\_IP\\_To\\_Device \(IPv4\)](#)
- [NU\\_Bind \(IPv4/IPv6\)](#)
- [NU\\_Bootp \(IPv4\)](#)
- [NU\\_Clear\\_Internal\\_Log](#)
- [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)
- [NU\\_Connect \(IPv4/IPv6\)](#)
- [NU\\_Delete\\_DNS\\_Server2 \(IPv4/IPv6\)](#)
- [NU\\_Delete\\_Host\\_Entry \(IPv4/IPv6\)](#)
- [NU\\_Delete\\_Route2 \(IPv4/IPv6\)](#)
- [NU\\_Detach\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Device\\_Up \(IPv4/IPv6\)](#)
- [NU\\_Dhcp \(IPv4\)](#)
- [NU\\_Dhcp\\_Release \(IPv4\)](#)
- [NU\\_Ethernet\\_Link\\_Down](#)
- [NU\\_Ethernet\\_Link\\_Up](#)

- [NU\\_Fcntl \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Check \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Init \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Reset \(IPv4/IPv6\)](#)
- [NU\\_FD\\_Set \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Next\\_Route \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Next\\_Route\\_Entry](#)
- [NU\\_Find\\_Route\\_By\\_Gateway \(IPv4/IPv6\)](#)
- [NU\\_Find\\_Socket \(IPv4/IPv6\)](#)
- [NU\\_Free\\_Host\\_Entry \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Default\\_Route \(IPv4\)](#)
- [NU\\_Get\\_Default\\_TTL \(IPv4\)](#)
- [NU\\_Get\\_DHCP\\_DUID \(IPv4/IPv6\)](#)
- [NU\\_Get\\_DHCP\\_IAID \(IPv4/IPv6\)](#)
- [NU\\_Get\\_DNS\\_Servers2 \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Domain\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Host\\_By\\_Addr \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Host\\_By\\_Name \(IPv4\)](#)
- [NU\\_Get\\_Host\\_MX](#)
- [NU\\_Get\\_Host\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Forwarding \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Node\\_By\\_Addr \(IPv4/IPv6\)](#)
- [NU\\_Get\\_IP\\_Node\\_By\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Peer\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Get\\_PMTU \(IPv4/IPv6\)](#)
- [NU\\_Get\\_Reasm\\_Max\\_Size \(IPv4\)](#)
- [NU\\_Get\\_Sock\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Getsockopt \(IPv4/IPv6\)](#)
- [NU\\_Getsockopt\\_IP\\_BROADCAST\\_IF \(IPv4\)](#)

- `NU_Getsockopt_IP_HDRINCL` (IPv4)
- `NU_Getsockopt_IP_MULTICAST_IF` (IPv4)
- `NU_Getsockopt_IP_MULTICAST_TTL` (IPv4)
- `NU_Getsockopt_IP_RECVIFADDR` (IPv4)
- `NU_Getsockopt_IP_TOS` (IPv4)
- `NU_Getsockopt_IP_TTL` (IPv4)
- `NU_Getsockopt_SO_BROADCAST` (IPv4)
- `NU_Getsockopt_SO_KEEPALIVE` (IPv4/IPv6)
- `NU_Getsockopt_SO_LINGER` (IPv4/IPv6)
- `NU_Getsockopt_SO_RCVBUF` (IPv4/IPv6)
- `NU_Getsockopt_SO_REUSEADDR` (IPv4/IPv6)
- `NU_Getsockopt_TCP_CFG_DSACK`
- `NU_Getsockopt_TCP_CFG_SACK`
- `NU_Getsockopt_TCP_CONGESTION_CTRL`
- `NU_Getsockopt_TCP_DELAY_ACK`
- `NU_Getsockopt_TCP_FIRST_RTO`
- `NU_Getsockopt_TCP_FIRST_TIMEOUT`
- `NU_Getsockopt_TCP_KEEPALIVE_R2`
- `NU_Getsockopt_TCP_KEEPALIVE_WAIT`
- `NU_Getsockopt_TCP_MAX_PROBES`
- `NU_Getsockopt_TCP_MAX_R2`
- `NU_Getsockopt_TCP_MAX_RTO`
- `NU_Getsockopt_TCP_MAX_SYN_R2`
- `NU_Getsockopt_TCP_MSL`
- `NU_Getsockopt_TCP_NODELAY` (IPv4/IPv6)
- `NU_Getsockopt_TCP_PROBE_TIMEOUT`
- `NU_Getsockopt_TCP_RCV_WINDOWSIZE`
- `NU_Getsockopt_TCP_SND_WINDOWSIZE`
- `NU_Getsockopt_TCP_TIMESTAMP`

- [NU\\_Getsockopt\\_TCP\\_WINDOWSCALE](#)
- [NU\\_Getsockopt\\_UDP\\_NOCHECKSUM \(IPv4\)](#)
- [NU\\_IF\\_FreeNameIndex \(IPv4/IPv6\)](#)
- [NU\\_IF\\_IndexToName \(IPv4/IPv6\)](#)
- [NU\\_IF\\_NameIndex \(IPv4/IPv6\)](#)
- [NU\\_IF\\_NameToIndex \(IPv4/IPv6\)](#)
- [NU\\_Inet\\_NTOP \(IPv4/IPv6\)](#)
- [NU\\_Inet\\_PTON \(IPv4/IPv6\)](#)
- [NU\\_Init\\_Devices \(IPv4/IPv6\)](#)
- [NU\\_Init\\_Net \(IPv4/IPv6\)](#)
- [NU\\_Ioctl \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_FIONREAD \(IPV4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCDARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFDSTADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGIFNETMASK \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGETVLAN \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCGETVLANPRIO \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCGHWCAP \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCICMPLIMIT](#)
- [NU\\_Ioctl\\_SIOCIFREQ \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCSARP \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#)
- [NU\\_Ioctl\\_SIOCSIFADDR \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSETVLAN \(IPv4\)](#)
- [NU\\_Ioctl\\_SIOCSETVLANPRIO \(IPv4/IPv6\)](#)
- [NU\\_Is\\_Connected \(IPv4/IPv6\)](#)
- [NU\\_Listen \(IPv4/IPv6\)](#)

- [NU\\_Ping \(IPv4\)](#)
- [NU\\_Ping2 \(IPv4/IPv6\)](#)
- [NU\\_Push \(IPv4/IPv6\)](#)
- [NU\\_Rarp \(IPv4\)](#)
- [NU\\_Recv \(IPv4/IPv6\)](#)
- [NU\\_Recvmsg \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_From \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_From\\_Raw \(IPv4/IPv6\)](#)
- [NU\\_Recv\\_IF\\_Addr \(IPv4\)](#)
- [NU\\_Remove\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Remove\\_IP\\_From\\_Device \(IPv4/IPv6\)](#)
- [NU\\_Rip2\\_Initialize \(IPv4\)](#)
- [NU\\_Select \(IPv4/IPv6\)](#)
- [NU\\_Send \(IPv4/IPv6\)](#)
- [NU\\_Sendmsg \(IPv4/IPv6\)](#)
- [NU\\_Send\\_To \(IPv4/IPv6\)](#)
- [NU\\_Send\\_To\\_Raw \(IPv4/IPv6\)](#)
- [NU\\_Set\\_Default\\_TTL \(IPv4\)](#)
- [NU\\_Set\\_Device\\_Hostname](#)
- [NU\\_Set\\_DHCP\\_DUID \(IPv4/IPv6\)](#)
- [NU\\_Set\\_DHCP\\_IAID \(IPv4/IPv6\)](#)
- [NU\\_Set\\_Domain\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Set\\_Host\\_Name \(IPv4/IPv6\)](#)
- [NU\\_Set\\_IP\\_Forwarding \(IPv4/IPv6\)](#)
- [NU\\_Set\\_MDNS\\_Hostname\\_Callback](#)
- [NU\\_Set\\_Reasm\\_Max\\_Size \(IPv4\)](#)
- [NU\\_Shutdown \(IPv4/IPv6\)](#)
- [NU\\_Socket \(IPv4/IPv6\)](#)
- [NU\\_Setsockopt \(IPv4/IPv6\)](#)

- `NU_Setsockopt_IP_ADD_MEMBERSHIP` (IPv4)
- `NU_Setsockopt_IP_DROP_MEMBERSHIP` (IPv4)
- `NU_Setsockopt_IP_BROADCAST_IF` (IPv4)
- `NU_Setsockopt_IP_HDRINC` (IPv4)
- `NU_Setsockopt_IP_MULTICAST_IF` (IPv4)
- `NU_Setsockopt_IP_MULTICAST_TTL` (IPv4)
- `NU_Setsockopt_IP_RECVIFADDR` (IPv4)
- `NU_Setsockopt_IP_TOS` (IPv4)
- `NU_Setsockopt_IP_TTL` (IPv4)
- `NU_Setsockopt_SO_BROADCAST` (IPv4)
- `NU_Setsockopt_SO_KEEPALIVE` (IPv4/IPv6)
- `NU_Setsockopt_SO_LINGER` (IPv4/IPv6)
- `NU_Setsockopt_SO_RCVBUF` (IPv4/IPv6)
- `NU_Setsockopt_SO_REUSEADDR` (IPv4/IPv6)
- `NU_Setsockopt_TCP_CFG_DSACK`
- `NU_Setsockopt_TCP_CFG_SACK`
- `NU_Setsockopt_TCP_CONGESTION_CTRL`
- `NU_Setsockopt_TCP_DELAY_ACK`
- `NU_Setsockopt_TCP_FIRST_PROBE_TIMEOUT`
- `NU_Setsockopt_TCP_FIRST_RTO`
- `NU_Setsockopt_TCP_KEEPALIVE_R2`
- `NU_Setsockopt_TCP_KEEPALIVE_WAIT`
- `NU_Setsockopt_TCP_MAX_PROBES`
- `NU_Setsockopt_TCP_MAX_R2`
- `NU_Setsockopt_TCP_MAX_RTO`
- `NU_Setsockopt_TCP_MAX_SYN_R2`
- `NU_Setsockopt_TCP_MSL`
- `NU_Setsockopt_TCP_PROBE_TIMEOUT`
- `NU_Setsockopt_TCP_NODELAY` (IPv4/IPv6)



- `NU_Setsockopt_TCP_RCV_WINDOWSIZE`
- `NU_Setsockopt_TCP_TIMESTAMP`
- `NU_Setsockopt_TCP_WINDOWSCALE`
- `NU_Setsockopt_UDP_NOCHECKSUM (IPv4)`
- `NU_Setup_Configured_Tunnel (IPv6)`
- `NU_Socket_Connected`
- `NU_Start_mDNS_Query`
- `NU_Stop_mDNS_Query`
- `NU_Test_Link_Up`
- `NU_Update_Route (IPv4/IPv6)`
- `NU_Update_DNS_Record`
- `NU_ZC_Allocate_Buffer (IPv4/IPv6)`
- `NU_ZC_Deallocate_Buffer (IPv4/IPv6)`
- `NU_ZC_Recv (IPv4/IPv6)`
- `NU_ZC_Recv_From (IPv4/IPv6)`
- `UM_Add_User`
- `UM_Del_User`
- `UM_Delete_User_Permissions`
- `UM_Find_User`
- `UM_Find_User_First`
- `UM_Find_User_Next`
- `UM_Get_User_Count`
- `UM_Validate_User`

## NET Shell Commands

The following commands can be issued from a Nucleus Shell command terminal (for example serial terminal, telnet, and so on).

- `ipconfig`

## ipconfig

Provides information about networking interfaces. This command allows for the discovery of basic networking interface configuration and status.

### Usage

```
ipconfig
```

### Arguments

- None

### Return Values

- Information about target networking interfaces, including current status and configuration.

### Examples

```
> ipconfig

loopback
  Status:                UP
  Hardware Address:      00:00:00:00:00:00
  IPv6 Address:          Unavailable
  IPv4 Address:          127.0.0.1
  Subnet Mask:           255.0.0.0
  Default Gateway:       10.0.2.2

eth0
  Status:                UP
  Hardware Address:      52:54:00:12:34:56
  IPv6 Address:          Unavailable
  IPv4 Address:          10.0.2.15
  Subnet Mask:           255.255.255.0
  Default Gateway:       10.0.2.2
```

### Related Topics

[NET Shell Commands](#)

## Configuring Nucleus NET

This chapter contains advice to assist the developer in maximizing optimization in regards to throughput and code size.

The file *os/include/networking/net\_cf.h* contains all user-configurable macros for Nucleus NET.

## Metadata Options File

The following are the current network configuration options located in the *os/networking/net/metadata* file (“default” is the default option).

```
option("include_ip_fwd") {
    default false
    enregister false
    description "Include IP forwarding support in stack"
}

option("include_udp") {
    default true
    enregister false
    description "Include UDP socket support in stack"
}

option("include_tcp") {
    default true
    enregister false
    description "Include TCP socket support in stack"
}

option("include_congestion_ctrl") {
    default true
    enregister false
    description "Include TCP congestion control support in stack"
}

option("include_pmtu_discvry") {
    default true
    enregister false
    description "Include Path MTU Discovery support in stack"
}

option("include_sack") {
    default true
    enregister false
    description "Include TCP selective ACK support in stack"
}

option("include_dsack") {
    default true
    enregister false
    description "Include TCP duplicate selective ACK support in stack"
}

option("include_window_scale") {
    default true
    enregister false
    description "Include TCP window scale option support in stack"
}

option("include_timestamp") {
    default true
    enregister false
    description "Include TCP timestamp option support in stack"
}

option("include_lmt_d_tx") {
    default true
    enregister false
    description "Include TCP limited transmit support in stack"
}
```

```
}

option("include_ip_raw") {
    default      false
    enregister    false
    description "Include RAW IP support in stack"
}

option("include_ll_config") {
    default      false
    enregister    false
    description "Include IPv4 link-local autoconfiguration support in
stack"
}

option("include_hw_offload") {
    default      false
    enregister    false
    description "Include hardware offloading support in stack"
}

option("include_net_debug") {
    default      false
    enregister    false
    description "Include advanced debugging support in stack"
}

option("include_mdns") {
    default      false
    enregister    false
    description "Include the Multicast DNS protocol in the networking
stack."
}

option("num_mdns_q_elements") {
    description "Size of the mDNS event queue.  The default size of this
queue is 8.  If there will be a lot of continuous queries initiated
from the application, this value should be increased."
    default      8
    values       8..128
    hidden       false
}

option("mdns_tsk_prio") {
    default      25
    enregister    false
    description "Priority of the mDNS master task."
}

option("mdns_wake_tsk_prio") {
    default      25
    enregister    false
    description "Priority of the mDNS wake task."
}

option("mdns_master_tsk_size") {
    description "Size, in bytes, of the mDNS master task."
    default      2000
}
```

```
        values      512..8192
        hidden      false
    }

    option("mdns_wake_tsk_size") {
        description "Size, in bytes, of the mDNS wake task. This should not
            be set to less than the minimum stack size for the specific platform
            as defined by nu.os.kern.plus.core.min_stack_size. If it is, the
            value will be reset at compile-time."
        default      250
        values      250..8192
        hidden      false
    }

    option("mdns_signal") {
        description "The callback signal used to inform the callback task
            that a change has been made to a record of interest to the task, as
            specified by NU_Start_mDNS_Query()"
        default      31
        values      0..31
        hidden      false
    }

    option("dns_sd_default_ttl") {
        default      120
        enregister false
        description "Default value to use for TTL field of local PTR, SRV
            and TXT records created when using NU_DNS_SD_Register_Service."
    }

    option("dns_sd_default_prio") {
        default      0
        enregister false
        description "Default value to use for priority field of local SRV
            records created when using NU_DNS_SD_Register_Service."
    }

    option("dns_sd_default_weight") {
        default      0
        enregister false
        description "Default value to use for weight field of local SRV
            records created when using NU_DNS_SD_Register_Service."
    }
}
```

## Reduce the Footprint Size

You should be careful to scrutinize the file *os/include/networking/net\_cf.h* and include only those components of Nucleus NET that will be used by the final node.

For example, if you have no need for the node to use UDP, set the macro `INCLUDE_UDP` to `NU_FALSE` so the additional code is removed from the final image.

## Optimal Buffer Size

The maximum amount of data a Nucleus NET buffer can contain is configurable via the macro `NET_PARENT_BUFFER_SIZE` in *os/include/networking/net\_cf.h*. This macro specifies the number of bytes that can be stored in a single buffer in a buffer chain. For maximum efficiency through the stack, you should configure this parameter to the smaller of the largest packet that will be most often transmitted and the largest interface MTU in the system. The goal is to avoid multiple buffers per buffer chain so processes do not have to walk a chain of buffers.

For example, if the final node consists of an FTP Server which most often serves files of size greater than 1Kb over Ethernet, `NET_PARENT_BUFFER_SIZE` should be set to `ETHERNET_MTU`, because the FTP Server will put the maximum number of bytes in each packet up to the MTU (or MSS of the other side). However, if the FTP Server most often serves files of only 100 bytes, `NET_PARENT_BUFFER_SIZE` should be set to 100 + the length of the TCP Header (since FTP uses TCP).

## Multicasting

By disabling multicasting, the footprint of the final image will be reduced, and throughput will be increased for forwarded and received packets. You should consider whether multicasting is necessary for the node, and if not, disable the macro `INCLUDE_IP_MULTICASTING` in *os/include/networking/net\_cf.h*. Note that if the node is using IPv6, multicasting should not be disabled since Neighbor Discovery (IPv6 address resolution) relies on multicasting.

## Loopback Device

The loopback device is used to transmit and receive packets over the target node. By disabling loopback, the footprint of the final image will be reduced, and throughput will be increased for forwarded and received packets. You should consider whether loopback is necessary for the node, and if not, disable the macro `INCLUDE_LOOPBACK_DEVICE` in *os/include/networking/net\_cf.h*.

## ARP

The ARP protocol (Address Resolution Protocol) is used only by Ethernet devices. Therefore, if the target node will not contain an Ethernet device, the macro `INCLUDE_ARP` should be disabled in *os/include/networking/net\_cf.h*. This will reduce the footprint of the final image and increase throughput for forwarded and received packets.

## Zero Copy Interface

Nucleus NET provides a Zero Copy interface to increase efficiency through the stack. This interface should be used for an optimal sockets-based application.

Refer to the “Extended Discussion” chapter of this guide for further information on the Zero Copy interface.

## Specialized Routines

### Socket Options

The functions `NU_Setsockopt()` and `NU_Getsockopt()` allow you to set and retrieve the value of socket options. Since Nucleus NET provides you with numerous socket options via this API, the code for these routines is quite large. As a result, if you set only one socket option, the code for all socket options is compiled into the final image.

In response to this growing size, Nucleus NET provides separate routines for setting and getting each socket option. The routines are named as follows; `NU_Setsockopt_OPTNAME()` and `NU_Getsockopt_OPTNAME()`, where `OPTNAME` is the value of the `optname` parameter that would otherwise be passed into `NU_Setsockopt()` or `NU_Getsockopt()`.

For more information, see “[NET Overview](#)” on page 55.

### IOCTL Options

The function `NU_Ioctl()` allows you to perform special functions on an interface or other object. Since Nucleus NET provides you with numerous IOCTL options via this API, the code for this routine is quite large. As a result, if you set only one IOCTL option, the code for all IOCTL options is compiled into the final image.

In response to this growing size, Nucleus NET provides separate routines for each IOCTL option. The routines are named as follows; `NU_Ioctl_OPTNAME()`, where `OPTNAME` is the value of the `optname` parameter that would otherwise be passed into `NU_Ioctl()`.

For more information, see “[NET Overview](#)” on page 55.

## Optimized Checksum and Block Copy Routines

Nucleus NET provides a set of optimized block copy and checksum routines.

### Using the Optimized Block Copy Routine

The block copy routines are for 8-bit, 16-bit and 32-bit aligned hardware. These routines are located in the file *nbc.c* at *os/networking/net/optimizations/block\_copy*.

#### Procedure

To use the optimized routine, perform the following steps:

1. In the file *os/include/networking/target.h*, change the macro `NU_BLOCK_COPY` to `NU_Block_Copy`.
2. Add the file *os/networking/net/optimizations/block\_copy/nbc.c* to the build.
3. In the file *nbc.c*, set the appropriate bit alignment for the hardware to `NU_TRUE`.

**Table 2-45. Using the Optimized Block Copy Routine**

Macro	Description
<code>OPTIMIZE_8_BIT</code>	Enable the optimized block copy routine for 8-bit aligned hardware.
<code>OPTIMIZE_16_BIT</code>	Enable the optimized block copy routine for 16-bit aligned hardware.
<code>OPTIMIZE_32_BIT</code>	Enable the optimized block copy routine for 32-bit aligned hardware.

## Related Topics

[Optimized Checksum and Block Copy Routines](#)

[Inline Code](#)

## Using the Optimized Assembly Checksum Routine

The C-source code used to create the assembly version of the checksum routine is located at *os/networkgin/net/optimizations/checksum*. Currently, the assembly code for the optimized checksum routine is available only for the ARM (using RVCT tools) and Power PC (using Microtec tools) platforms. If you have an ARM or Power PC port, the optimized checksum routine will be located at *net/optimizations/checksum*. Additionally, an endian-dependent optimized C version of the checksum routine is available at *os/networking/net/optimizations* for platforms that do not have an assembly version of this routine. The assembly code will be located in a subdirectory named for your platform at *os/networking/net/optimizations/checksum*.

## Prerequisites

- Your port supports an optimized checksum routine.

## Procedure

1. In the file *os/include/networking/target.h*, change the macro `TCPCHECK_ASM` to `NU_TRUE`.
2. Add the assembly file containing the optimized checksum routine to your project.
3. Rebuild the code



## Related Topics

[Optimized Checksum and Block Copy Routines](#)

[Inline Code](#)

## Using the Optimized Endian-Dependent C Checksum Routine

If your port does not support an optimized assembly version of the checksum routine, you can use the optimized endian-dependent C checksum routine located at the root of the directory *optimizations/checksum*.

### Procedure

To use the optimized C routine, follow these steps:

1. Copy the file at *os/networking/net/optimizations* into the *net/src* directory, overwriting the existing file.
2. If the hardware is big endian, uncomment the macro `TLS_TC_BIG_ENDIAN` located at the top of the file.
3. Rebuild the Nucleus NET library as usual.

## Related Topics

[Optimized Checksum and Block Copy Routines](#)

[Using the Optimized Assembly Checksum Routine](#)

## Inline Code

When a routine is made inline, the function call is replaced with the actual function body in all places where the inlined function is called. A significant increase in throughput has been observed during testing by making certain routines inline within Nucleus NET and Ethernet drivers. Because inline is not ANSI standard, this code cannot be made inline in the code base of Nucleus NET, therefore it must be done by you. Refer to your toolset manual for the specific syntax to use for inline routines.

## Nucleus NET Inline Routines

The following routines in Nucleus NET are recommended to be made inline;  
`DEV_Get_Dev_For_Vector()`, `DEV_Recover_TX_Buffers()`, `EightZeroTwo_Input()`,  
`EightZeroTwo_Output()`, `IP_Forward()`.

## Ethernet Driver Inline Routines

It is recommended that all Ethernet routines called from another Ethernet routine be made inline. For example, if the Ethernet transmission HISR calls a separate transmission routine, the transmission routine should be made inline to avoid the function call.

## Ethernet Drivers

### Receive and Transmit Buffer Descriptors

You should perform throughput tests to determine the optimal number of receive and transmit buffer descriptors to use. For example, the Nucleus NET networking group performed the same throughput test with a certain Nucleus Ethernet driver using 64, 128, and 256 buffer descriptors. From the test results, 128 buffers produced better throughput than 64 buffers, but 256 buffers produced worse performance than 128 buffers. Therefore, 128 buffers was determined optimal for this driver.





































































































































































































































































































































































































































































































































































































































## NET User Overview

This chapter describes how to create simple client-server applications for both IPv4 and IPv6 networks using Nucleus OS. The following modules of Nucleus OS will be discussed here:

- Nucleus NET (*nucleus\os\networking\net*)
- Nucleus IPv6 (*nucleus\os\networking\netv6*)

Nucleus NET is an implementation of a comprehensive TCP/IP stack optimized for embedded applications. In order to provide both compatibility with existing standards and interfaces and at the same time provide a means to operate the network stacks efficiently, Nucleus NET provides for a socket API that is modeled after Berkeley Sockets. Nucleus NET also includes full support for IPv6 thereby enabling users to develop next generation applications for the IPv6 infrastructures. Besides providing for an optimized (for both size and performance), robust and a scalable dual IPv4/IPv6 stack implementation, Nucleus NET also:

- Provides for a Zero-Copy interface for throughput optimization.
- Supports Zero Configuration networking for M2M applications through the latest features added to Nucleus NET: mDNS and DNS-SD.
- Is Phase-II certified as “IPv6 Ready” to ensure full RFC compliance and interoperability.

This chapter demonstrates the ease with which you can develop sophisticated multi-threaded networking applications for your devices using Nucleus OS.

## Operation

This section contains information about:

- [Initialization](#)
- [Use Cases](#)

## Initialization

The networking stack (Nucleus NET) gets initialized automatically (at Run-level 7) during the initialization sequence of the operating system. Please refer to the [Initialization](#) chapter in the *Nucleus Kernel Guide* for more information on Run-level initialization sequence. The following

components of the OS must successfully initialize first in order for Nucleus NET initialization to be successful:

- Kernel (Nucleus PLUS)
- Device Manager
- Registry Service

In order to successfully use all the services provided by Nucleus NET, at least one of the Network devices must also become available.

## Use Cases

This section offers details about:

- [Using Standard TCP](#)
- [Using Standard UDP](#)
- [Using Nucleus NET for IPv6 Networking](#)
- [Optimizing the Throughput Through the Use of Zero-copy Interface](#)
- [Using Blocking Services in the Networking Stack](#)
- [Operations on Nucleus NET Devices](#)
- [Input/Output Control \(IOctl\) Support in Nucleus NET](#)

## Using Standard TCP

One important decision for using TCP communication is to choose whether your application will act as a server or as a client. A "server" usually runs persistently and listens for connections from "clients". In fact, a server is capable of simultaneously accepting connections from multiple clients. On the other hand, a "client" is an entity which connects to a server for data exchange. A TCP connection is a stream of data which connects a server with a client. The sections below describe how to write a TCP server or a TCP client depending on what role your user application performs.

## Prerequisites

- Configuration

The following configuration option must be enabled in the Nucleus Configuration Editor (UI editor) for this use case:

```
nu.os.net.stack.include_tcp = true
```

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

- Initialization

Please see the [Initialization](#) section.

## Procedure for a Standard TCP Server

1. Create a socket of the appropriate family type for the communications, use [NU\\_Socket \(IPv4/IPv6\)](#).
2. Bind the Server port and IP address, use [NU\\_Bind \(IPv4/IPv6\)](#).
3. Perform a Listen call to establish the number of simultaneous requests that can be queued, use [NU\\_Listen \(IPv4/IPv6\)](#).
4. Perform an Accept service call to ‘wait’ for client connections, use [NU\\_Accept \(IPv4/IPv6\)](#).

After the Accept service call returns (for example, a connection has been established) data can be transferred (control returns from [NU\\_Accept \(IPv4/IPv6\)](#)).

5. Transfer the data, use [NU\\_Send \(IPv4/IPv6\)](#) and [NU\\_Recv \(IPv4/IPv6\)](#).
6. Close the connection, use [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)

## Procedure for a Standard TCP Client

1. Create a socket of the appropriate family type for the communications [NU\\_Socket \(IPv4/IPv6\)](#).
2. Perform a connect service call, use [NU\\_Connect \(IPv4/IPv6\)](#).

After the connect service call returns (for example, a connection has been established) data can be transferred (control returns from [NU\\_Connect \(IPv4/IPv6\)](#)).

3. Transfer the data, use [NU\\_Send \(IPv4/IPv6\)](#) and [NU\\_Recv \(IPv4/IPv6\)](#).
4. Close the connection, use [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)

## Related Topics

[Use Cases](#)

[Using Standard UDP](#)

## Using Standard UDP

UDP is inherently connection less. However, it is expected that in general, both sides of a data transfer are aware of the transfer, and port numbers have been predefined. The operation for a sender and receiver only differ in the service calls that are made. That is, both the sending and

receiving of data require a full address specification of the recipient and sender respectively. Data transfer over UDP is simply performed by using the appropriate interfaces specified in the [NET](#) chapter. UDP can operate in both server and client mode. Additionally, it can be used to communicate from one machine to another without the notion of a server. However, in order to receive a Datagram, a port must be specified, and the application must have issued a [NU\\_Recv\\_From \(IPv4/IPv6\)](#) service call before the sender actually sends the packet in order for the UDP to process the datagram.

## Prerequisites

- Configuration

The following configuration option must be enabled in the Nucleus Configuration Editor (UI editor) for this use case:

```
nu.os.net.stack.include_udp = true
```

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

- Initialization

Please see the [Initialization](#) section.

## Procedure for a Standard UDP Server

1. Create a socket of the appropriate family type for the communications, use [NU\\_Socket \(IPv4/IPv6\)](#).
2. Bind the local server address to the socket, use [NU\\_Bind \(IPv4/IPv6\)](#).
3. Wait on for client's data, use [NU\\_Recv\\_From \(IPv4/IPv6\)](#).
4. Send data - optional, use [NU\\_Send\\_To \(IPv4/IPv6\)](#).
5. Close the socket, use [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)

## Procedure for a Standard UDP Client

1. Create a socket of the appropriate family type for the communications, use [NU\\_Socket \(IPv4/IPv6\)](#).
2. Send data - optional, use [NU\\_Send\\_To \(IPv4/IPv6\)](#).
3. Receive data, use [NU\\_Recv\\_From \(IPv4/IPv6\)](#).
4. Close the socket, use [NU\\_Close\\_Socket \(IPv4/IPv6\)](#)

## Related Topics

[Use Cases](#)

[Using Standard TCP](#)

## Using Nucleus NET for IPv6 Networking

Nucleus NET is dual stack meaning it can support both IPv4 and IPv6 simultaneously. Compatibility between IPv4 and IPv6 nodes is critical for a successful transition to IPv6 and hence the dual stack approach enables nodes to communicate over both IPv4 and IPv6 simultaneously. However, Nucleus NET can be configured to communicate over only IPv4, only IPv6 or IPv4 and IPv6 via configuration settings described below.

Using Nucleus NET, the steps for creating applications for either IPv4 or IPv6 or both are not much different from each other. The discussion below details the differences:

### Family Types

A family type is associated with an IP address and a socket. For IPv4 the family type is `NU_FAMILY_IP`. When IPv6 support is desired, the family type must be `NU_FAMILY_IP6`. The family type `NU_FAMILY_IP6` refers to an IPv6 address, and an IPv4/IPv6 socket. A socket created with the family `NU_FAMILY_IP6` can accept and transmit packets over IPv4 or IPv6. From the application's point of view, all communications over a socket created using `NU_FAMILY_IP6` are with IPv6 nodes. As an example, if IPv6 is enabled and the application opens a new connection (maybe as a web server) with a socket created using `NU_FAMILY_IP6`, the application serves both IPv4 and IPv6 clients. If an IPv4 client connects, the `NU_Accept` call would return a `client_addr` with the family parameter set to `NU_FAMILY_IP6`, and the address is an IPv4-Mapped IPv6 address.

### IPv4-Mapped IPv6 Addresses

An IPv4-Mapped IPv6 address contains the IPv4 address in the lower 32 bits of the IPv6 address in network byte order. If a socket is created using `NU_FAMILY_IP6`, all IPv4 addresses are returned to the application as IPv4-Mapped IPv6 Addresses with family type of `NU_FAMILY_IP6`.

When porting an IPv4 application to add support for IPv6, some of the calls to the sockets API will need to be modified to indicate whether the connection is regarding an IPv4, IPv6, or any

family of node. Calls to the following API services must be modified according to the information in [Table 3-1](#):

**Table 3-1. Calls to be Modified for IPv6**

API Service	Changes required
<a href="#">NU_Get_Host_By_Name (IPv4)</a>	The new function to use is <a href="#">NU_Get_Host_Name (IPv4/IPv6)</a> . Set the <b>family</b> parameter to the family of the host to retrieve. If retrieving an IPv4 host, set the <b>family</b> to NU_FAMILY_IP. If retrieving an IPv6 host, set the <b>family</b> to NU_FAMILY_IP6. The <b>flags</b> parameter is used only in conjunction with the <b>family</b> type NU_FAMILY_IP6. If you created an IPv6 socket and are going to transmit data to the address returned by the <a href="#">NU_Get_Host_Name (IPv4/IPv6)</a> call, set the <b>flags</b> parameter to DNS_V4MAPPED. This returns all IP addresses in the form of IPv4-Mapped IPv6 addresses.
<a href="#">NU_Get_Host_By_Addr (IPv4/IPv6)</a>	Set the <b>len</b> parameter according to the <b>family</b> of the host you are trying to retrieve. <ul style="list-style-type: none"><li>• If retrieving an IPv4 host, set <b>len</b> to IP_ADDR_LEN.</li><li>• If retrieving an IPv6 host, set <b>len</b> to IP6_ADDR_LEN.</li></ul>
<a href="#">NU_Socket (IPv4/IPv6)</a>	Calls to this function must be modified to pass in the correct <b>family</b> type. <ul style="list-style-type: none"><li>• To create an IPv4-specific socket, pass in NU_FAMILY_IP.</li><li>• To create an IPv6/IPv4 socket, pass in NU_FAMILY_IP6.</li></ul>
<a href="#">NU_Bind (IPv4/IPv6)</a>	The <b>myaddr</b> parameter of the call must be modified to fill in the <b>family</b> type of the address to which you are binding. <ul style="list-style-type: none"><li>• If binding to an IPv4 address, pass in NU_FAMILY_IP.</li><li>• If binding to an IPv6 address, set the <b>family</b> type to NU_FAMILY_IP6.</li><li>• If binding on an IPv6 socket to the Unspecified Address (all zeros), set the <b>family</b> type to NU_FAMILY_IP6.</li><li>• If binding to an IPv4 socket to the Unspecified Address (all zeros) set the <b>family</b> type to NU_FAMILY_IP.</li></ul> Note that the address to which you bind must be of the same <b>family</b> as the socket.
<a href="#">NU_Send_To (IPv4/IPv6)/NU_Send_To_Raw (IPv4/IPv6)</a>	The <b>to</b> parameter of the call must be modified to fill in the <b>family</b> type of the address to which you are sending data. <ul style="list-style-type: none"><li>• If sending data to an IPv4 node on an IPv4 socket, set the <b>family</b> type to NU_FAMILY_IP.</li><li>• If sending data to an IPv4 node on an IPv6 socket, set the <b>family</b> type to NU_FAMILY_IP6 and use the IPv4-Mapped IPv6 Address returned from the DNS inquiry.</li><li>• If sending to an IPv6 node, set the <b>family</b> type to NU_FAMILY_IP6.</li></ul>



Table 3-1. Calls to be Modified for IPv6 (cont.)

API Service	Changes required
<a href="#">NU_Sendmsg (IPv4/IPv6)</a>	<p>The <b>msg_name</b> parameter of the call must be modified to fill in the <b>family</b> type of the address to which you are sending data.</p> <ul style="list-style-type: none"> <li>• If sending data to an IPv4 node on an IPv4 socket, set the <b>family</b> type to <code>NU_FAMILY_IP</code>.</li> <li>• If sending data to an IPv4 node on an IPv6 socket, set the <b>family</b> type to <code>NU_FAMILY_IP6</code> and use the IPv4-Mapped IPv6 Address returned from the DNS inquiry.</li> <li>• If sending to an IPv6 node, set the <b>family</b> type to <code>NU_FAMILY_IP6</code>.</li> </ul>
<a href="#">NU_Connect (IPv4/IPv6)</a>	<p>The <b>servaddr</b> parameter of the call must be modified to fill in the <b>family</b> type of the address to which you are connecting.</p> <ul style="list-style-type: none"> <li>• If connecting to an IPv4 node on an IPv4 socket, set the <b>family</b> type to <code>NU_FAMILY_IP</code>.</li> <li>• If connecting to an IPv4 node on an IPv6 socket, set the <b>family</b> type to <code>NU_FAMILY_IP6</code> and use the IPv4-Mapped IPv6 address returned from the DNS inquiry.</li> <li>• If connecting to an IPv6 node, set the <b>family</b> type to <code>NU_FAMILY_IP6</code>.</li> </ul>
<a href="#">NU_Find_Socket (IPv4/IPv6)</a>	<p>The <b>local_addr</b> and <b>foreign_addr</b> parameters of the call must be modified to fill in the <b>family</b> type of the addresses using the socket.</p> <ul style="list-style-type: none"> <li>• If the addresses using the socket are IPv4 addresses, set the <b>family</b> type to <code>NU_FAMILY_IP</code>.</li> <li>• If the addresses using the socket are IPv6 addresses, set the <b>family</b> type to <code>NU_FAMILY_IP6</code>.</li> </ul>

## Prerequisites

- Configuration

To add IPv6 support in applications, the "netv6" component of the Nucleus OS must be enabled in the Nucleus Configuration Editor (UI editor):

```
1) nu.os.net.ipv6.enable = true
```

Enabling this component adds IPv6 support to the Networking stack. Besides this, IPv6 must also be enabled for each networking device that will use IPv6. The following configuration setting serves as an example for how to enable IPv6 on a particular network device:

```
2) ${platform}.ethernet0.mw_settings.eth_enable_ipv6 = true
```

In this example, IPv6 is being enabled on the networking device "ethernet0" for the particular platform in use.

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

- Initialization

Please see the [Initialization](#) section.

## Related Topics

[Use Cases](#)

[Optimizing the Throughput Through the Use of Zero-copy Interface](#)

## Optimizing the Throughput Through the Use of Zero-copy Interface

Nucleus NET provides a Zero-copy interface to increase efficiency through the stack. This interface can be used as an alternative to the standard sockets API in an optimal sockets-based application. The Zero-copy Interface improves performance by eliminating the need for the Transport Layer to copy data into Nucleus NET buffers when sending and receiving data. The application copies the data directly into Nucleus NET buffers and passes the buffers to the Transport Layer for transmission. Zero-copy services are currently available for sending and receiving data using TCP and UDP sockets.

## Procedure

1. Allocate buffer, use [NU\\_ZC\\_Allocate\\_Buffer \(IPv4/IPv6\)](#)
2. Deallocate buffer, use [NU\\_ZC\\_Deallocate\\_Buffer \(IPv4/IPv6\)](#)
3. Find the total number of bytes of data in the provided buffer chain, use [NU\\_ZC\\_BUF\\_LEN \(IPv4/IPv6\)](#)
4. Find the number of bytes in the provided segment of a buffer chain, use [NU\\_ZC\\_SEGMENT\\_BYTES\\_AVAIL \(IPv4/IPv6\)](#)
5. Get a pointer to the next segment in a chain of Nucleus NET buffers, use [NU\\_ZC\\_SEGMENT\\_NEXT \(IPv4/IPv6\)](#)
6. Get a pointer to the beginning of the data section of a segment in a chain of Nucleus NET buffers, use [NU\\_ZC\\_SEGMENT\\_DATA \(IPv4/IPv6\)](#)
7. Find the number of bytes that can be copied into the segment use [NU\\_ZC\\_SEGMENT\\_BYTES\\_LEFT \(IPv4/IPv6\)](#)

## Related Topics

[Use Cases](#)

[Using Blocking Services in the Networking Stack](#)

## Using Blocking Services in the Networking Stack

Nucleus NET provides for both blocking and non-blocking services. All services block by default. If non-blocking services are required, Nucleus NET provides the [NU\\_Fcntl \(IPv4/IPv6\)](#) service to change the blocking mode of the socket. Non-blocking services return immediately whether the request is completed or not. Therefore, it is possible in a non-blocking read request that no bytes will be returned. Read services normally block when data is not available to receive and are resumed once data is available to pass to the application task. Write services can block, for example, when there are not enough resources (buffer space) available to send the data. The task is resumed when resources become available and the write request can be satisfied. Both active connection requests and passive connection requests can be made either blocking or non-blocking. When a write request is made, the task may assume that the data will be transferred to the specified recipient (unless an error was detected by the protocol stack, for example, an incorrect socket identifier was specified).

### Procedure

This procedure shows the steps for changing the blocking mode:

Blocking mode is the default condition for all I/O socket services. It is often desirable though to determine if data is available from the network without blocking execution of the requesting task. For such cases, Nucleus NET provides a mechanism to specify whether or not the reads are to be blocked. This capability is provided through the [NU\\_Fcntl \(IPv4/IPv6\)](#) service call. This service can be used to specify that subsequent reads on the indicated socket are to be blocked or not.

As an example, if it is desired that a read operation be performed without blocking, take the following steps:

1. Call the routine [NU\\_Fcntl \(IPv4/IPv6\)](#) to set a flag on the socket with the value of:
  - NU\_FALSE to indicate that the subsequent read call is not to block.
  - NU\_BLOCK to indicate that the subsequent read call is to block.
2. Perform a read from the socket.

### Related Topics

[Use Cases](#)

[Operations on Nucleus NET Devices](#)

## Operations on Nucleus NET Devices

There are several Application-Layer services that can be performed on Nucleus Networking Devices.

- IP addresses can be added and removed.
- Devices can be added or completely removed from the system.

## Methods for Adding an IPv4 Address to a Device

There are three ways to add an IPv4 address to an interface as shown in [Table 3-2](#):

**Table 3-2. Adding an IPv4 Address to a Device**

Method	Procedure
Manual Configuration	<p>A Nucleus NET device can have multiple simultaneous IPv4 addresses. You can add IPv4 address to devices using the <a href="#">NU_Attach_IP_To_Device (IPv4)</a> API by passing in the following parameters:</p> <ul style="list-style-type: none"><li>• <b>name</b>: The unique name of the device on which to add the IPv4 address.</li><li>• <b>ip_addr</b>: The new IPv4 address to add to the device in decimal notation.</li><li>• <b>subnet</b>: The subnet mask of the new IPv4 address to add to the device.</li></ul>
Dynamic Configuration using DHCP (use <a href="#">NU_Dhcp (IPv4)</a> ), BOOTP (use <a href="#">NU_Bootp (IPv4)</a> ) or RARP	<p>Nucleus NET offers three forms of dynamic configuration for IPv4 addresses - DHCP, BOOTP and RARP. Each of these three methods requires a complementary server on the local subnet to handle the respective request.</p>
Automatic Configuration using Dynamic Autoconfiguration	<p>Nucleus NET supports automatic configuration of an IPv4 addresses for use on the local network, similar to the Stateless Address Autoconfiguration feature offered by Nucleus IPv6.</p>

## Methods for Adding an IPv6 Address to a Device

There are three ways to add an IPv6 address to an interface, as shown in [Table 3-3](#):

**Table 3-3. Adding an IPv6 Address to a Device**

Method	Procedure
Manual Configuration	<p>You can add an IPv6 address to the devices using the API <a href="#">NU_Add_IP_To_Device (IPv6)</a> by passing in the following parameters:</p> <ul style="list-style-type: none"><li>• <b>dev_name</b>: The unique name of the device on which to add the IPv6 address.</li><li>• <b>ip_addr</b>: The new IPv6 address to add to the device in hexadecimal notation.</li><li>• <b>prefix_length</b>: The prefix length of the IPv6 address to add. This is logically equivalent to the IPv4 subnet mask in that the prefix length specifies the number of significant bits in the IPv6 address.</li><li>• <b>preferred_lifetime</b>: The preferred lifetime of the new address. When this time expires, the address will not be used for any new communications, however, the address will continue to be used for existing communications. The user should specify 0xffffffff for an infinite preferred lifetime.</li><li>• <b>valid_lifetime</b>: The valid lifetime of the new address. When this time expires, the address will be removed from the device. The user should specify 0xffffffff for an infinitely valid lifetime</li></ul>
Dynamic Configuration using <a href="#">NU_Dhcp6 (IPv6)</a>	Nucleus NET (with IPv6) users can obtain IPv6 addresses dynamically at run-time using DHCP over IPv6.
Automatic Configuration Stateless Address Auto configuration	Nucleus NET (with IPv6) supports Stateless Address Auto configuration by default to configure link-local addresses. If an IPv6-enabled router is present on the local link and able to serve globally unique IPv6 prefixes, the Nucleus IPv6 node is also able to configure its own globally unique IPv6 address.

## Removing an IP Address from a Device

Just as an address can be added, it can be removed from a Nucleus NET device. Both IPv4 and IPv6 addresses can be removed from devices using the [NU\\_Remove\\_IP\\_From\\_Device \(IPv4/IPv6\)](#) API by passing in the following parameters:

- **name**: The unique name of the device from which to remove the address.
- **ip\_addr**: The address to remove from the device.
- **family**: The family type of the address to remove.

## Removing a Nucleus NET Device from the System

In Nucleus NET, a device can be completely removed from the stack using the API [NU\\_Remove\\_Device \(IPv4/IPv6\)](#) by passing in the following parameters:

- **name:** The unique name of the device to remove.
- **flags:** This parameter is currently unused.

### Related Topics

[Use Cases](#)

[Input/Output Control \(IOctl\) Support in Nucleus NET](#)

## Input/Output Control (IOctl) Support in Nucleus NET

Nucleus NET provides for the IOctl service through the [NU\\_Ioctl \(IPv4/IPv6\)](#) API. This API allows you to perform special functions on a Networking device or other objects. Nucleus NET provides with numerous IOctl options via this API. For optimization purposes, Nucleus NET also provides separate routines for each of the IOctl option. The routines are named as `NU_Ioctl_OPTNAME()`, where `OPTNAME` is the value of the **optname** parameter that would otherwise be passed into [NU\\_Ioctl \(IPv4/IPv6\)](#). The actions that can be performed using these APIs are shown in [Table 3-4](#):

**Table 3-4. IOctl Actions**

Actions	OPTNAME to be used
Get the IP address associated with an interface	SIOCGIFADDR
Get the IP address on the foreign side of a PPP link. Only valid for PPP links.	SIOCGIFDSTADDR
Set the MAC address associated with an interface.	SIOCSPHYSADDR
Set the IP address associated with the interface.	SIOCSIFADDR
Add or modify an entry in the ARP cache	SIOCSARP
Delete an entry from the ARP cache	SIOCDEARP
Get an entry from the ARP cache	SIOCGARP
Get the IPv6 address associated with an interface.	SIOCGIFADDR_IN6
Get the IPv6 address on the foreign side of a PPP link. Only valid for PPP links	SIOCGIFDSTADDR_IN6
Change the VLAN ID association for device	SIOCSETVLAN
Get the VLAN ID association for device	SIOCGETVLAN
Request hardware offloading features	SIOCGHWCAP
Set hardware Offloading Mode	SIOCShwOPTS

**Table 3-4. IOCTL Actions (cont.)**

Actions	OPTNAME to be used
Change the VLAN priority association for device	SIOCSETVLANPRIO
Get the VLAN priority association for device	SIOCGETVLANPRIO
Get the subnet mask associated with an IP address	SIOCGIFNETMASK
Set the ICMP error rate-limiting interval	SIOCICMPLIMIT

## Related Topics

[Use Cases](#)[Using Blocking Services in the Networking Stack](#)

# Nucleus NET Internals

This is a detailed description of:

- [Threads](#)
- [Buffer Management](#)
- [Events Dispatcher](#)
- [Interface Management](#)
- [Error Logging](#)
- [Support for Hardware Offloading](#)

## Threads

Users of the Nucleus NET protocol stack access networking services from a task context. That is, when a Nucleus NET service is requested, it executes in the context of the requesting task.

[Table 3-5](#) summarizes all the OS tasks that get created as part of the Nucleus NET initialization:

**Table 3-5. Nucleus Threads**

Task/ISR	Buffer Suspension High Priority ISR	Events Dispatcher	NET Demux
Name	buf_susp	EvntDisp	TIMER
Entry Function	MEM_Buffer_Suspension_HI SR()	NU_EventsDispatcher()	NET_Demux()
Type	HISR	Task	Task
Priority	2	3	3

**Table 3-5. Nucleus Threads (cont.)**

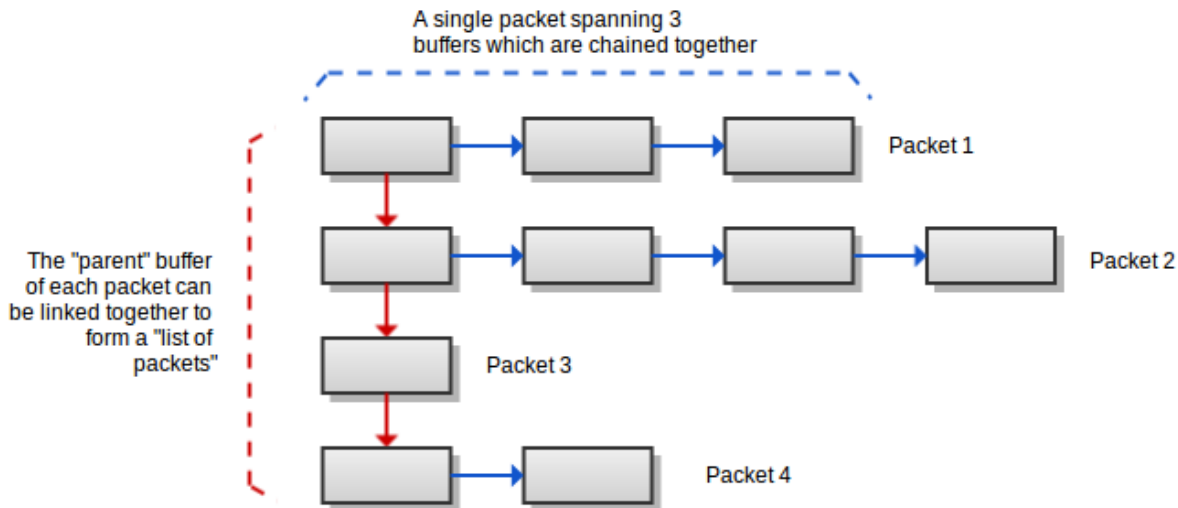
Task/ISR	Buffer Suspension High Priority ISR	Events Dispatcher	NET Demux
Stack Size	CFG_NU_OS_NET_STACK_BUF_HISR_STACK_SIZE <b>nu.os.net.stack.buf_hisr_stack_size</b>	CFG_NU_OS_NET_STACK_EV_STACK_SIZE <b>nu.os.net.stack.ev_stack_size</b>	CFG_NU_OS_NET_STACK_TM_STACK_SIZE <b>nu.os.net.stack.tm_stack_size</b>
Description	When buffers become available this HISR wakes up all the tasks that are suspended waiting for buffers.	This task processes all events generated by the stack. An example of such an event would be the retransmission of a TCP packet.	This task processes all the received packets.
Notes	The default stack size for this task is 512 bytes. Setting this stack too small can have dire consequences (stack overflow, etc). Setting this stack size too large, simply wastes memory (RAM).	The default stack size for this task is 4000 bytes. Since this task is responsible for all networking stack's timing related events, setting this stack too small can have dire consequences (stack overflow, etc). Setting this stack size too large, simply wastes memory (RAM).	The default stack size for this task is 5000 bytes. Since this task handles processing of all the received packets, setting this stack too small can have dire consequences (stack overflow, etc). Setting this stack size too large, simply wastes memory (RAM).

## Buffer Management

The buffers within Nucleus NET hold all the data that is transmitted and received to and from the network. These buffers are used from the Link Layer all the way up to the Application Layer (when using Zero-Copy buffers). All buffers are created upon initialization of the network stack. Specialized routines are used to “allocate” and “deallocate” buffers in the system.

Nucleus NET uses fixed-size buffers which can be chained together to provide storage for a packet which exceeds the size of a single buffer. For example, if we have buffers of size 500 bytes each then a packet of 1500 bytes may span 3 such buffers. Furthermore, another dimension is added to this chain when we need to maintain a list of "packets" because each packet can span multiple buffers. This is shown in [Figure 3-1](#).



**Figure 3-1. Buffer Management**

To allow creation of such two dimensional lists, each buffer of Nucleus NET contains two pointers. One is the "next\_buffer" pointer which is used to link together all buffers of a single packet (as shown in blue in the diagram) and the second is the "next" pointer which is used to maintain a "list of packets" by including only the "head" or "parent" buffers in the list.

## Buffer Creation

All buffers available to the networking stack are created upon initialization at the call to [NU\\_Init\\_Net \(IPv4/IPv6\)](#). Buffers cannot be dynamically created or deallocated after initialization. The configuration options can be modified to customize buffer quantity and size within the networking stack.

- **nu.os.net.stack.max\_bufs:** This option specifies the number of buffers that will be created at initialization. If the network interface driver maintains a cache of network buffers, the user must keep this in mind when configuring the number of buffers within the system.
- **nu.os.net.stack.buf\_size:** This option specifies the size of each buffer. It is senseless to set this parameter to a value larger than the maximum packet size that can be transmitted on a link. For an Ethernet device, this maximum size is 1536. Since this is the maximum size, no buffer larger than this size will ever be created; therefore, any additional memory created for the buffer would be wasted.

## Buffer Data Structure

When transmitting data, buffers are chained together to form a buffer chain. Each chain is comprised of one parent buffer, and zero or more non-parent buffers.

- The parent buffer holds extra information (meta-data) about the buffer chain that is necessary to decipher what is contained in the buffer chain. This information contains:
  - a pointer to its free buffer list
  - the total length of the buffer chain
  - some data specific to TCP (if this is a TCP packet)
- Non-parent buffers hold data only.

## Buffer Lists

Since all buffers are created upon initialization, various lists of the buffers are maintained within the networking stack to keep track of their current location. The two dimensional linking of buffers was explained at the start of this section. Therefore note that some of the lists mentioned below are single dimensional since they only deal with individual buffers whereas other lists are two dimensional because they need to keep track of "packets" instead of individual buffers.

### Free List

The Free List (**MEM\_Buffer\_Freelist**) contains all buffers that are not being used by any process. This list is where all buffers will be allocated from during networking procedures to send and receive data and where all buffers are deallocated to when the process is finished with the buffer. If this list is empty, data cannot be received or transmitted.

### Device Transmit Queue

The Device Transmit Queue is specific to a network device. It contains the buffers to be transmitted to the network through that device.

### Incoming Buffer List

The global list **MEM\_Buffer\_List** is used to pass incoming buffers from the Link Layer to the Network Layer via the system task **NET\_Demux**.

When a packet is received by a network interface at the Link Layer, the Link Layer reserves a chain of buffers from the network stack large enough to hold the entire payload of the packet, copies the data into the network buffer chain by a DMA transfer, a `memcpy()`, or some equivalent process, and places the buffer chain on the **MEM\_Buffer\_List** to be processed by the Network Layer. The event **Buffers\_Available** is then set by the Link Layer, and the system task **NET\_Demux**, which is suspended waiting for the **Buffers\_Available** event to be resumed, is awakened. **NET\_Demux** then invokes the proper Network Layer routine associated with the buffer chain to further process the buffers.

There can only be one instance of **NET\_Demux** in the system, and this routine must obtain the TCP/IP semaphore before continuing execution; therefore, buffers will build up on the **MEM\_Buffer\_List** waiting to be processed by the system.

## Socket Receive List

Each socket has a corresponding Socket Receive List (**s\_recvlist** of the **sock\_struct**) specific to that socket that holds incoming network buffers destined to an application. A chain of buffers is placed on the appropriate socket's receive list and delivered to the application by one of two methods, depending on whether the associated socket is a regular socket or Zero Copy socket. For non-Zero Copy sockets, when the Application makes a call to receive data (via [NU\\_Recv\\_From \(IPv4/IPv6\)](#), [NU\\_Recv \(IPv4/IPv6\)](#), etc), the data from the network buffer is copied into the user's buffer, and the network buffer is freed. Note that a user buffer is nothing more than a pointer to memory, not actually a specific data structure. For Zero Copy sockets, when the Application makes a call to receive data, a pointer to the data within the network buffer is returned to the application. The application is then responsible for freeing the network buffer. The network buffer remains on the socket receive list until the Application frees it.

## TCP Buffers List

The TCP module is very complex and can hold pointers to buffers on various lists. This section details those lists.

## TCP Retransmission List

Once a TCP packet has been transmitted, it must be acknowledged by the recipient. If the recipient fails to receive the packet, the sender must be able to retransmit the packet; therefore the packet must be held until it is acknowledged. The TCP Retransmission List will maintain a pointer to the chain of buffers containing the transmitted packet until acknowledgment for the data contained in the buffer chain has been received. Even when this buffer chain is on the device's transmit queue, the TCP Port will maintain a pointer to the buffer.

When the buffer chain is created in the routine `TCPSS_Net_Write()`, the buffer is placed onto the **out.packet\_list** for the port, and the buffer's **mem\_dlist** is set to `OS_NULL` to indicate to lower layers that this chain of buffers should not be moved onto any other list when freeing the chain.

Immediately before the packet is passed to the IP layer, interrupts are disabled and the `NET_TX_QUEUE` flag is set in the buffer. This flag ensures that TCP does not attempt to retransmit the packet or remove the packet from the **out.packet\_list** while the packet is still on the device's transmit queue from a previous transmission, and to ensure that the buffers are cleaned up if the TCP port is closed before the device has finished transmitting the packet.

## Out of Order Packet List

The TCP module places buffers on the respective socket's receive list in order according to the sequence number in the TCP header. When a packet is received that is not the next "expected"

packet according to the last sequence number received, this buffer chain will be stored on the Out of Order List for the port (**in.ooo\_list** of the port structure), awaiting arrival of the missing data.

## Nagle Packet List

When the Nagle Algorithm is enabled for a TCP socket, and you call **NU\_Send (IPv4/IPv6)** specifying a buffer of data smaller than the MSS for the connection, that buffer of data may not be immediately transmitted upon completion of the call to **NU\_Send (IPv4/IPv6)**. The reason for this is that it is less efficient to send multiple small buffers of data than a less number of large buffers of data, so the stack waits a certain amount of time “hoping” the application layer will call **NU\_Send (IPv4/IPv6)** again so more data can be added to the packet. A pointer to this buffer is held in the **out.nextPacket** field of the port structure until it is finally transmitted.

## IP Reassembly List

When the network layer receives a buffer of data that has been fragmented by the sending host, it must hold onto the incoming buffers until all fragmented packets are received so the data can be reassembled. These buffers are stored on the global list **IP\_Frag\_Queue**.

## Address Resolution List

When the Link Layer transmits data, it must first determine the link-layer address associated with the destination IP address of the packet. To do this, the Address Resolution Protocol (ARP) is used. This process involves transmitting and receiving a set of ARP packets to resolve the link-layer address. While the system is waiting for address resolution to complete, the buffer of outgoing data must be stored for safe keeping to be transmitted upon completion of the ARP process or freed upon failure. The global list **ARP\_Res\_List** is used to store these buffers.

## Buffer Management Routines

The buffer management routines are system level routines not to be used at the application layer. These routines are used within the network stack and by network interface drivers only.

## Buffer Creation

The internal system routine **MEM\_Init** is called from the API routine **NU\_Init\_Net (IPv4/IPv6)** to allocate all buffers available on the **MEM\_Buffer\_Freelist**. Upon successful completion of this routine, no more buffers will be allocated in the system. Furthermore, the memory for these buffers is never deallocated by any process in the system.

A total of MAX\_BUFFERS NET\_BUFFER structures will be allocated and added to the **MEM\_Buffer\_Freelist**. Memory for buffers is allocated on a strict 4-byte alignment to ensure all buffer structures are word-aligned.

The counter **MEM\_Buffers\_Used** is initialized to zero to indicate that all buffers in the system are available for use. This global counter is incremented as buffers are allocated by processes and decremented as they are deallocated.

## Buffer Allocation

The internal system routine **MEM\_Buffer\_Chain\_Dequeue** is used by the network stack and network interface driver to allocate a chain of buffers capable of holding a payload of an indicated length. The buffers are taken from the **MEM\_Buffer\_Freelist**, and the global counter **MEM\_Buffers\_Used** is decremented by the number of buffers in the chain. The **next\_buffer** parameter of each buffer in the chain is set up accordingly to link the buffers together in a proper chain. All other fields are initialized to zero. You are responsible for filling in the following parameters for every buffer chain allocated using this routine:

- **mem\_dlist** – The list onto which these buffers will be deallocated.
- **data\_ptr** – The beginning of the payload of the packet. This is set to **mem\_parent\_packet** initially and decremented / incremented as data is added or removed.
- **mem\_total\_data\_len** – The total number of bytes of data in the entire buffer chain. This should equal the sum of each **data\_len** parameter for each buffer in the chain.
- **data\_len** – The total number of bytes of data in each buffer in the chain.

## Buffer Deallocation

The following routines are used to deallocate buffers:

### MEM\_Buffer\_Chain\_Free

When a process is finished using a buffer chain, and all buffers in the chain are being deallocated to the same list, the routine **MEM\_Buffer\_Chain\_Free** is used to deallocate the buffers. The network stack uses the routine **MEM\_Buffer\_Chain\_Free** to return buffers to the specified **mem\_dlist** either when the buffers have been successfully processed or when an error occurs that requires the buffers to be deallocated.

### MEM\_Recover\_TX\_Buffers

When a network interface driver has finished transmitting a chain of buffers, it uses the routine **DEV\_Recover\_TX\_Buffers** to return the chain of buffers to the appropriate lists. Otherwise, the network interface driver uses **MEM\_Buffer\_Chain\_Free**.

### MEM\_Multiple\_Buffer\_Chain\_Free

When a process is finished using a buffer chain, and the buffers in the chain are being deallocated to the multiple lists, the routine **MEM\_Multiple\_Buffer\_Chain\_Free** is used to deallocate the buffers. For example, the ARP module uses the routine

**MEM\_Multiple\_Buffer\_Chain\_Free** to deallocate buffers when address resolution fails. The routine **DEV\_Recover\_TX\_Buffers** also uses this routine to deallocate buffers to the appropriate lists.

## Events Dispatcher

Events are used throughout Nucleus NET for various tasks; window probes, retransmissions, DHCP, ARP requests, and keeping TCP connections alive, to name a few. Many events will be pending processing at the same time. These events must be executed in an orderly fashion so that no event is lost. The routines discussed in this section are system level routines and not accessible by the Application Layer.

The Events Dispatcher schedules and handles events within Nucleus NET. The Events Dispatcher is a task that removes events from the Event List and the Event Queue, deciphers the event, and processes it accordingly. Events are placed on the Event List by the Timer task. Events are placed on the Event Queue by different networking processes.

Nucleus NET provides the ability to register events dynamically at Run-time via the system routine **EQ\_Register\_Event**. This ability removes the need for the Events Dispatcher to be modified each time a new event is created since the function callback routine for the specific event is created at Run-time. There are, however, some legacy hard-coded events still in the Events Dispatcher. These are described in the following [Table 3-6](#). Note that new events must use the dynamic event registration routine instead of hard-coding the Events Dispatcher with events.

**Table 3-6. Events Description**

Event	Description
CONOPEN	Connection has opened, CONCLASS.
CONCLOSE	The other side has closed its side of the connection.
CONFAIL	Connection open attempt has failed.
UNUSED_EVENT_SPOT	This used to be CONRX, but this has been removed.
CONNUL	Just a null event, used to wake up the dispatcher from a indefinite sleep should a timer queue entry be posted. It could be used for other purposes also.
UDPDATA	UDP data has arrived on listening port, USERCLASS.
TCPRETRANS	TCP segment retransmission event
WINPROBE	Window Probe event
TCPACK	TCP ACK transmission event
CONTX	Buffer needs to be sent
SELECT	TCP select time-out event

Table 3-6. Events Description (cont.)

Event	Description
APPRESOLVE	ARP event
RARP_REQUEST	A ARP request event
EV_IGMP_REPORT	Send an IGMP Group Report
EV_IP_REASSEMBLY	Timeout for reassembly of an IPv4 datagram
EV_IP6_REASSEMBLY	Timeout for reassembly of an IPv6 datagram
ICMP_ECHO_TIMEOUT	Timeout waiting for a ping reply
IPDATA	IP data has arrived
TCPCLOSETIMEOUTSFW2	Timeout the closing of a connection
NAT_CLEANUP_TCP	NAT specific event
NAT_CLENUUP_UDP	NAT specific event
NAT_TIMEOUT	NAT specific event
NAT_CLOSE	NAT specific event
NAT_CLEANUP_ICMP	NAT specific event
MEM_RESUME	Resume a task suspended on memory allocation

## Setting Timers

Timer events are set by the network processes on events that should not be executed immediately, but rather after some interval of time. The network processes do this by calling **TQ\_Timerset()** which sets a timer for an event. This event is then placed on the Event List **EQ\_Event\_List**.

The Event List is kept in due time order. This certifies the next event to expire will be at the front of the list. Since it is possible for multiple events to be due at the same time, it is necessary to organize the list to prevent it from becoming too long. Each event on the Event List has a “duplist”. The “duplist” is a list of duplicate events (events that have matching due times). This shorter list reduces the time to insert new events onto the Event List.

It is possible that before the timer interval finishes, a network process determines that the event is unnecessary. A process can cancel a timer event using the routine **TQ\_Timerunset()**. This function will then remove the event from the Event List.

## Setting Events

The setting of an event adds that event to the Event Queue, **eQueue**, which causes the event to be handled by the Events Dispatcher after all expired timer events have been processed.



## Registering Events

As networking processes are executed, it may be necessary to dynamically add to the list of possible events. This can be done by registering a new event. To register a new event, the event handler is passed to the registration function, and the macro value associated with that event is passed back to the caller. Whenever an event is set with the system, the macro value returned must be used to refer to the registered event.

## Handling Events

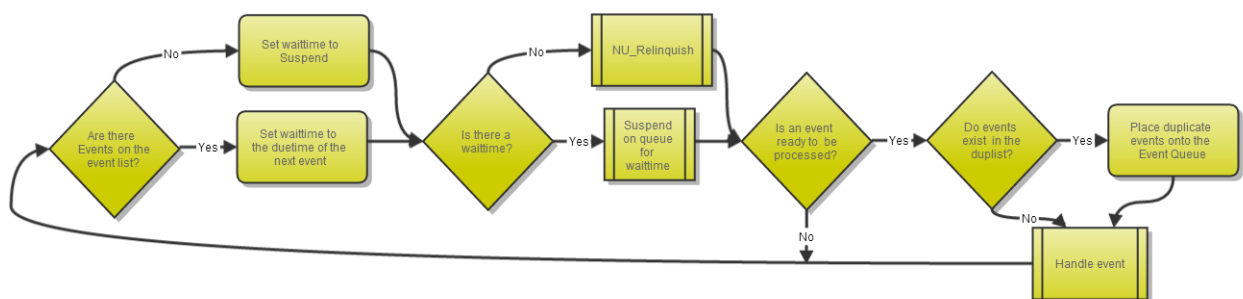
The handling of events is a continuous loop the Event Dispatcher must follow. It begins with checking the Event List. Recall that the Event List is populated by **TQ\_TimerSet()**. If events exist within the Event List, the Events Dispatcher must check the due times of these events. If an event is due, the variable waittime is set to **NU\_NO\_SUSPEND**. If an event is not due, wait time is set to the earliest due time. If an event does not exist, wait time is set to **NU\_SUSPEND**.

At this point it is required that the Events Dispatcher relinquish control to the system to prevent system starvation. If an event is due, the dispatcher will call **NU\_Relinquish()**. If the wait time is anything other than **NU\_NO\_SUSPEND**, an event is not ready to be processed, and the dispatcher will suspend on the Event Queue for the amount of time specified by wait time, or until an event is received.

If an event exists, the dispatcher will begin processing it. Many timer events can expire at the same time, however, the dispatcher can only address them one at a time. To handle each of these events, the dispatcher will check the “duplist”. If events are present on the “duplist”, each one will be placed on the Event Queue, to be handled after the current event has been processed.

Now, the Events Dispatcher will determine what type of event has been set, and all the appropriate event handler. With the event handler completed, the Events Dispatcher is ready to process the next event, if one exists.

**Figure 3-2. Event Dispatcher Loop**





## Interface Management

This section explains the interaction between the Nucleus OS and the Nucleus Networking packet components.

### Interaction between components (System View)

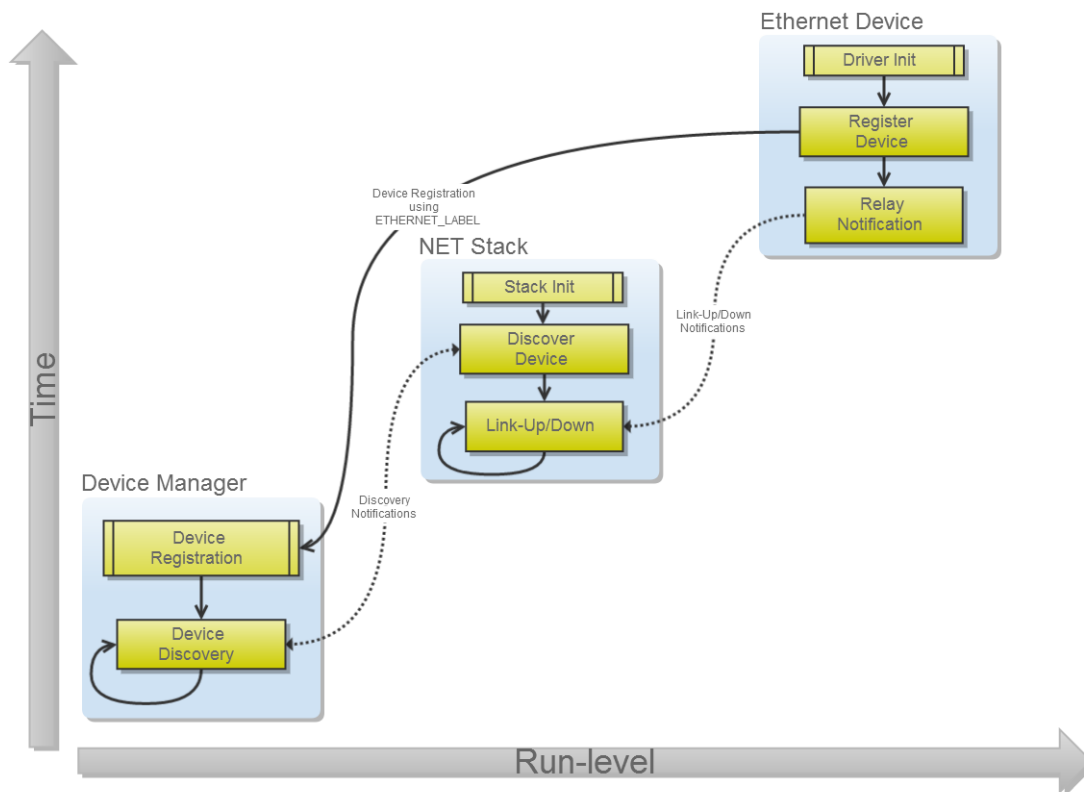
The various components of the Operating System that Nucleus Networking component interacts with, comes up at various stages of the run-level initialization sequence. During run-level initialization, each of these components initialize all of their respective data structures and threads of execution and get ready to interact with each other to effectively manage reception and transmission of network traffic to and from the device. Following is the sequence in which this initialization and subsequent interaction takes place:

1. The Device Manager component gets initialized first and as part of this initialization, the following things happen:
  - a. All the data structures and the framework required to register and un-register Nucleus devices gets initialized.
  - b. A task called the Device Discover task **DevDisc** is spawned that listens for any [un]registration events and sends appropriate notifications to the concerned components.
2. The Nucleus Networking Stack component gets initialized next and as part of this initialization, the following things happen:
  - a. All the data structures and the framework required to perform TCP/IP communications get initialized. This also includes initialization of NET buffers.
  - b. A task to monitor when a particular network device comes up **Link-Up** or goes down **Link-Down** called the Link Status task (NETLINK) is spawned.
3. The Ethernet Device Driver component gets initialized in the end and it performs the following actions:
  - a. All the data structures and the framework required to add/register a new ethernet device to the Operating System get initialized.
  - b. A framework to relay various notifications with regards to the device's status is also setup.
4. After the Ethernet Device driver component is finished with the initial setup, it registers the Ethernet device with the Device Manager using the ETHERNET\_LABEL.
5. The Device Manager, upon the successful registration of the ethernet device, notifies the Networking stack that a new device has been registered / discovered.

- Once the Networking stack discovers the new device, it initiates the **Link-Up** process with the Device and subsequently receives a **Link-Up** notification and then it continues with TCP/IP operations.

This sequence is described in [Figure 3-3](#):

**Figure 3-3. Interface Management**



## Error Logging

Nucleus NET provides a logging module to log informational and error messages from within the stack using the function **NLOG\_Error\_Log()**. Errors are also logged from within network drivers using the thread-safe function **NERRS\_Log\_Error()**.

## Message Format

Messages are stored in the global array **NLOG\_Entry\_List[]** in the following format:

```
_timeFromBoot_ ***ERROR***:  
_message_ _taskName_ File: _fileName_ (_lineNumber_) stat:  
_severity_
```

Where **\_severity\_** is one of the following:

- Informational
- Recoverable
- Severe
- Fatal

## Accessing Logging Information

You can access logging information by either examining the global array **NLOG\_Entry\_List[]** or configuring Nucleus NET to output logging information in real-time to a UDP server.

## Support for Hardware Offloading

In the interest of speed and to meet increasing industry demand, Nucleus NET supports various hardware offloading features. The following features are currently supported:

- Hardware Checksum Offloading of the IPv4
- Transport Layer Header Checksums for incoming and outgoing packets

In order to use hardware offloading features, those features must exist on the particular hardware.

## Hardware Checksum Offloading

Hardware checksum offloading moves the computation of the IPv4 and/or transport layer header checksums of outgoing packets and/or verification of the IPv4 and/or transport layer header checksums of incoming packets into the hardware. Since hardware is significantly faster than software, this greatly increases throughput.

The IPv6 header does not contain a checksum; therefore, there is no option for IPv6 header hardware checksum offloading.

## Configuration Options

The following configuration options enables/disables the ability to perform hardware checksum offloading on a device:

```
nu.os.net.stack.include_hw_offload
```

This option is disabled by default.

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

## Enabling Hardware Checksum Offloading

You can enable hardware checksum offloading features for a particular device at run-time using the routine [NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#) or specifying SIOCSHWOPTS in the call to [NU\\_Ioctl \(IPv4/IPv6\)](#). You must specify which hardware checksum offloading feature(s) to enable as in [Table 3-7](#):

**Table 3-7. Hardware Checksum Offloading Options**

Option	Purpose
HW_TX_IP4_CHKSUM	Enable hardware checksum offloading of the IP checksum in outgoing IPv4 packets.
HW_RX_IP4_CHKSUM	Enable hardware checksum offloading of the IP checksum in incoming IPv4 packets.
HW_TX_TCP_CHKSUM	Enable hardware checksum offloading of the TCP checksum in outgoing IPv4 packets.
HW_RX_TCP_CHKSUM	Enable hardware checksum offloading of the TCP checksum in incoming IPv4 packets.
HW_TX_TCP6_CHKSUM	Enable hardware checksum offloading of the TCP checksum in outgoing IPv6 packets.
HW_RX_TCP6_CHKSUM	Enable hardware checksum offloading of the TCP checksum in incoming IPv6 packets.
HW_TX_UDP_CHKSUM	Enable hardware checksum offloading of the UDP checksum in outgoing IPv4 packets.
HW_RX_UDP_CHKSUM	Enable hardware checksum offloading of the UDP checksum in incoming IPv4 packets.
HW_TX_UDP6_CHKSUM	Enable hardware checksum offloading of the UDP checksum in outgoing IPv6 packets.
HW_RX_UDP6_CHKSUM	Enable hardware checksum offloading of the TCP checksum in incoming IPv6 packets.
HW_TX_VLAN	Enable hardware offloading for transmission of VLAN packets.
HW_RX_VLAN	Enable hardware offloading for reception of VLAN packets.

## Disabling Hardware Checksum Offloading

Each time the routine is invoked to configure hardware checksum offloading features, the combination of features specified will be enabled, and all other routines will be disabled; therefore, you can disable a certain hardware checksum offloading feature by invoking

[NU\\_Ioctl\\_SIOCSHWOPTS \(IPv4/IPv6\)](#) or specifying SIOCSHWOPTS in the call to [NU\\_Ioctl \(IPv4/IPv6\)](#) and excluding the desired feature(s) to disable.

The following sections detail how it works in the stack:

## Device Structure Members

The internal DV\_DEVICE\_ENTRY device structure has been modified to include two new members specific to hardware offloading; **dev\_phy\_hw\_options\_enabled** and **dev\_phy\_hw\_options**. **dev\_phy\_hw\_options\_enabled** contains the hardware offloading features enabled by the user on the device. **dev\_phy\_hw\_options** contains the hardware offloading features supported by the device. This value is filled in by the driver-specific initialization routine at initialization.

## Outgoing Data

When data is transmitted, the checksum field of the respective header (TCP/UDP/IPv4) is zeroed out. If the `HARDWARE_OFFLOAD` macro is enabled, the code checks whether the respective hardware checksum offloading feature is enabled in the **dev\_hw\_options\_enabled** member of the device structure. If it is set, the checksum is not computed.

## Incoming Data

When data is received, if the `HARDWARE_OFFLOAD` macro is enabled, the code checks whether the respective hardware checksum offloading feature is enabled in the **dev\_hw\_options\_enabled** member of the device structure. If it is set, the checksum is not verified.

## Related Topics

[Nucleus NET Internals](#)

[Appendix](#)

# Appendix

This section includes:

- [Configuration Options](#)

## Configuration Options

Nucleus NET is highly configurable. It can be very easily configured as per application requirement to maximize optimization in regards to throughput and code size. [Table 3-8](#) lists the currently available configuration options, their usage and their possible values. When Nucleus OS is reconfigured or built for the first time, two files, *current.imageconfig* and *current.imageconfig.html* are created. These two files can be used to understand and customize a configuration.

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

**Table 3-8. NET Configuration Options**

Option	Description	Values
include_ip_fwd	Include IP forwarding support in stack	false
include_udp	Include UDP socket support in stack	true
include_tcp	Include TCP socket in stack	true
include_congestion_ctrl	Include TCP congestion support in stack	true
include_pmtu_discvry	Include Patch MTU Discovery support in stack	true





































# Chapter 4

## Network Address Translator (NAT)

---

### **Warning**



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## NAT Overview

The IP Network Address Translator (NAT) protocol is a router protocol that allows nodes on a private network to transparently communicate with nodes on an external network and vice versa. Nodes on a private network have not been assigned a globally unique IP address; therefore, communication with the external network would otherwise be impossible. This transparent communication is accomplished by modifying the IP and protocol-specific headers of packets flowing to and from the private network. Nucleus NAT solves three common problems with growing networks; shortage of globally unique IP addresses, firewall-like protection for the private network, and flexibility of network administration.

There are a variety of flavors of NAT. Nucleus NAT is a combination of the Network Address Port Translator protocol (NAPT) and Bi-Directional NAT.

NAPT solves the problem of a shortage of globally unique IP addresses by allowing all nodes on a private network to communicate with the external network by sharing a single external IP address. This is accomplished by replacing the IP address and TCP/UDP port number of the private node with an external IP address and TCP/UDP port number in the IP and TCP/UDP header. The number of simultaneous sessions is limited only by the number of available ports.

Bi-Directional NAT enables connections to be initiated from hosts on the external network as well as the private network. Specific ports on the NAT router are mapped to services on a private node via a portmap service. The NAT router relays all matching requests from the external network to the specific private node. This enables nodes on a private network to be configured as servers accessible by the external network.

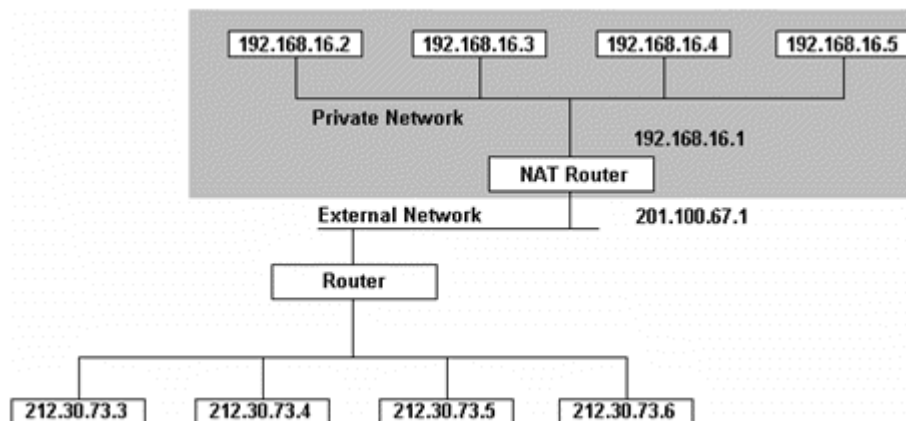
Since all connections must be initiated from the private network or registered with the portmap service, Nucleus NAT provides firewall-like protection for the private network. An intruder would have to first gain access to the NAT router in order to infiltrate the private network.

No modifications need to be made to the NAT router when a new node is added or existing nodes are removed or reconfigured. This provides flexibility of network administration.

The requirements for a NAT router are outlined in RFC 1631 and clarified in RFC 2663.

## Example NAT Router

**Figure 4-1. Example NAT Router**



In [Figure 4-1](#), the private network 192.168.16.x is hidden from the external network behind a NAT router. The NAT router has one external device (201.100.67.1) used to communicate with the external network and to protect the anonymity of the private nodes. The NAT router has one private device (192.168.16.1) used to communicate with the private network.

When a private node sends a packet to the external network, the NAT router intercepts the packet and replaces all instances of the private source IP address and TCP/UDP source port with the external IP address (201.100.67.1) and an assigned external TCP/UDP source port.

When an external node responds to a private node or initiates an acceptable connection with a private node, the NAT router intercepts the packet and replaces all instances of the external destination IP address (201.100.67.1) and assigned external destination TCP/UDP port with the private IP address and destination TCP/UDP port.

## Multiple Private Devices

In the example provided, the NAT router has only one private device. You specify which device(s) will be used to communicate with the private network(s) and which device will be used to communicate with the external network. Note that while only one device may be used to communicate with the external network, multiple devices can be used to communicate with multiple independent private networks. If a device exists that has not been specified as either private or external, NAT will ignore all packets transmitted on this device. For more information, see the [“NAT Functions”](#) on page 1022.



## Limited Resources

Nucleus NAT has access to limited resources as configured by you. You specify the number of simultaneous TCP, UDP, and ICMP connections that may be serviced by NAT. Once this maximum number of connections is reached, Nucleus NAT must deny any new connection requests of the respective protocol type and will return an ICMP error message to the invoking node as follows: If a node on the private network is attempting to make a new connection, an ICMP Source Quench message will be sent to the private node. If a node on the external network is attempting to make a new connection, an ICMP Port Unreachable message will be sent to the external node.

## NAT Protocol

The following table describes the level of protocol support provided by Nucleus NAT.

**Table 4-1. NAT Protocol**

Protocol	Support	Explanation
IP	YES	For packets outbound from the private network, NAT replaces the source IP address in the IP header with the external IP address. For packets inbound from the external network, NAT replaces the destination IP address in the IP header with the internal IP address. Adjustments are made checksum to the IP.
TCP	YES	For packets outbound from the private network, NAT replaces the source port number in the TCP header with the external port number. For packets inbound from the external network, NAT replaces the destination port number in the TCP header with the internal port number. Adjustments are made to the TCP checksum.
UDP	YES	For packets outbound from the private network, NAT replaces the source port number in the UDP header with the external port number. For packets inbound from the external network, NAT replaces the destination port number in the UDP header with the internal port number. Adjustments are made to the UDP checksum.
DNS	YES	See UDP.
HTTP	YES	See IP.
Telnet Client	YES	See TCP.
Telnet Server	YES	See TCP.

**Table 4-1. NAT Protocol (cont.)**

Protocol	Support	Explanation
TFTP Client	NO	Nucleus NAT does not support TFTP client in order to ensure the security of the private network. When a private node sends a TFTP request to an external TFTP server, the server does not answer on a well-known port, but opens a new port for communication. Since NAT needs to know this port number to verify the validity of the external node, the TFTP server's response will be rejected.
TFTP Server	YES	See UDP.
FTP Client	YES	Aside from the modifications made as described above for TCP, the FTP Application Level Gateway (ALG) provides the ability to perform PORT and PASV commands from the private network.
FTP Server	YES	See TCP.
ICMP	YES	The ICMP Application Level Gateway (ALG) provides the translation of data embedded within ICMP error messages.
DHCP Client	NO	Nucleus NET does not support relay switching for DHCP requests; therefore, a private node cannot query a DHCP server outside the private network.
DHCP Server	NO	Nucleus NET does not currently provide a DHCP server solution.
SNMP	NO	External SNMP managers cannot query private nodes. An SNMP ALG can be implemented to accomplish this task.

## NAT Fragmentation and Reassembly

Nucleus NAT supports the fragmentation and reassembly of packets to and from the private and external network.

## NAT Files

The following table defines the location and purpose of each file in the Nucleus NAT distribution:

**Table 4-2. NAT Files**

Distribution Directory	File Name	Purpose	Description
/	<i>nat_cfg.h</i>	Configuration	User-configurable macros.
/inc	<i>alg_defs.h</i>	Support	Data structure and macro definitions for the Application Level Gateways.

Table 4-2. NAT Files (cont.)

Distribution Directory	File Name	Purpose	Description
	<i>alg_extr.h</i>	Support	Function definitions for the Application Level Gateways.
	<i>nat_defs.h</i>	Support	Data structure and macro definitions for NAT.
	<i>nat_extr.h</i>	Support	Function definitions for NAT.
	<i>nat_mgmt.h</i>	Support	Function definitions for the statistics gathering functionality in Nucleus NAT.
/src	<i>alg.c</i>	ALG	Source code for the Application Level Gateways.
	<i>alg_icmp.c</i>	ICMP ALG	Source code for the ICMP Application Level Gateway.
	<i>alg_ftp.c</i>	FTP ALG	Source code for the FTP Application Level Gateway.
	<i>nat.c</i>	NAT	Source code for NAT.
	<i>nat_mgmt.c</i>	Statistics Gathering	Source code to retrieve statistics about the NAT module.
	<i>nat_nmi.c</i>	Initialization	Source code to automatically initialize the Nucleus NAT module from Nucleus PLUS using the NMI module.
	<i>nat_prtmap.c</i>	Portmapper	Source code to add and remove portmap entries within the NAT module.
	<i>nat_shutdown.c</i>	Uninitialization	Source code to shut down the Nucleus NAT module.

## NAT Data Structures

This section contains commonly used NAT data structures:

- [NAT\\_DEVICE](#)
- [NAT\\_PORTMAP\\_SERVICE](#)

### NAT\_DEVICE

The NAT\_DEVICE data structure is used to hold the names of all devices to be used on the internal network. NAT\_DEVICE is the typedef for the \_NAT\_DEVICE data structure:

```
typedef struct _NAT_DEVICE
{
    CHAR    nat_dev_name[DEV_NAME_LENGTH];
} NAT_DEVICE;
```

This array contains an entry for each internal device.

## Related Topics

[NAT Data Structures](#)

[NAT\\_Initialize](#)

## NAT\_PORTMAP\_SERVICE

NAT\_PORTMAP\_SERVICE is the typedef for the \_NAT\_PORTMAP\_SERVICE data structure described below. It is an entry for the Portmap Table.

```
typedef struct _NAT_PORTMAP_SERVICE
{
    UINT8          nat_internal_source_ip[4];
    UINT8          nat_protocol;
    UINT16         nat_internal_source_port;
    UINT16         nat_external_source_port;
} NAT_PORTMAP_SERVICE;
```

The members of this structure are described in [Table 4-3](#). Some members of this structure have been omitted as they are used internally.

**Table 4-3. NAT\_PORTMAP\_SERVICE**

Member	Description
nat_internal_source_ip	IP address of the internal node in the private network.
nat_protocol	Protocol to be used, can be one of: <ul style="list-style-type: none"><li>• IPPROTO_TCP</li><li>• IPPROTO_UDP.</li></ul>
nat_internal_source_port	The port number on which the internal node will be accessible from the node in the external network.
nat_external_source_port	The port number on the external node from which the internal node will be accessible.

## Related Topics

[NAT Data Structures](#)

[NAT\\_Portmap](#)

## NAT Functions

This chapter describes the API functions for Nucleus NAT.

- [NAT\\_Initialize](#)
- [NAT\\_Portmap](#)
- [NAT\\_Shutdown](#)

## NAT\_Initialize

Initializes the NAT data structures and timer tasks.

### Usage

```
STATUS NAT_Initialize (NAT_DEVICE *nat_internal_devices,  
                      INT          dev_count,  
                      CHAR          *nat_external_dev_name);
```

### Arguments

- **nat\_internal\_devices**  
Pointer to an [NAT\\_DEVICE](#) structure of size `dev_count` containing one entry for each internal device.
- **dev\_count**  
The number of internal devices.
- **nat\_external\_dev\_name**  
Pointer to the name of the device that will be used to communicate with the external network.

### Return Values

- **NU\_SUCCESS**  
NAT was successfully initialized.
- **NU\_NO\_MEMORY**  
NAT was not initialized due to a lack of memory.
- **NU\_TIMEOUT**  
A timeout occurred on the service.
- **NU\_POOL\_DELETED**  
The memory pool was deleted during suspension.

### Description

This service must be called after Nucleus NET and all devices have been initialized. Nucleus NAT does not currently get initialized by the OS. It must be explicitly initialized from the user's

application. The following are the steps involved in initializing Nucleus NAT on a device from an application:

Wait for the Networking support (networking stack and the networking interfaces) to be available. This can be done by using the API [NETBOOT\\_Wait\\_For\\_Network\\_Up](#).

Once the networking support is available on the device, a default route must be added to enable the nodes on the private network to communicate with the nodes on the public network. This can be accomplished by using the API [NU\\_Add\\_Route \(IPv4\)](#).

Once the appropriate routes are in place, invoke `NAT_Initialize()` with the appropriate parameters.

---

**Note**

Passing the correct parameters into `NAT_Initialize` requires the knowledge of the names of the network interfaces (both internal and external). Please refer to `current.imageconfig.html` located under the `:/output/toolchain/<platform>/debug` directory of your system project.

---

## Example

```
STATUS          status;
NAT_DEVICE      NAT_Devices[NAT_INTERNAL_DEVICES];

. . .

status = NETBOOT_Wait_For_Network_Up(NU_SUSPEND);

if (status == NU_SUCCESS)
{
    /* Copy the name of the internal device into the NAT_DEVICE data
     * structure.  If multiple internal devices are being used, each
     * of the device names must be copied into the successive elements
     * of the NAT_DEVICE data structure.
     */
    strcpy(NAT_Devices[0].nat_dev_name, "eth0");

    /* Initialize NAT with the NAT_DEVICE data structure, the number
     * of internal devices and the name of the external device.
     */
    status = NAT_Initialize(NAT_Devices, 1, "net0");
}

. . .
```

## Related Topics

[NAT Functions](#)

[NAT Data Structures](#)

## NAT\_Portmap

Register or unregister a private node as being accessible from the external network on the given port number.

### Usage

```
STATUS NAT_Portmap (NAT_PORTMAP_SERVICE *new_portmap_service,  
                   UINT8                  function);
```

### Arguments

- `new_portmap_service`  
A pointer to the [NAT\\_PORTMAP\\_SERVICE](#) data structure containing the specifications of the new service.
- `function`  
The specification to either add the node or delete the node - either `NAT_PORTMAP_ADD` or `NAT_PORTMAP_DELETE`.

### Return Values

- `NU_SUCCESS`  
The node was successfully registered or unregistered.
- `NU_INVALID_PARM`  
One of the parameters passed in to the function is invalid.
- `NU_NO_MEMORY`  
There is not enough memory to register the node.
- `NAT_ENTRY_EXISTS`  
The entry already exists.

### Example

```
STATUS status;  
UINT8 int_ip[4]= {192, 168, 16, 1};  
NAT_PORTMAP_SERVICE nat_port_map;  
  
. . .  
  
/* Populate "nat_port_map" structure. */  
memcpy(nat_port_map.nat_internal_source_ip, int_ip, 4);  
nat_port_map.nat_protocol = IPPROTO_TCP;  
nat_port_map.nat_internal_source_port = 80;  
nat_port_map.nat_external_source_port = 80;  
  
/* Add a portmap entry to the Portmap table, this results in  
 * port 80 of node 192.168.16.1 becoming accessible to the  
 * external nodes via TCP and hence the external nodes can open  
 * a connection with this port.  
 */
```



```
status = NAT_Portmap(nat_port_map, NAT_PORTMAP_ADD);
```

. . .

## Related Topics

[NAT Functions](#)

[NAT Data Structures](#)

## NAT\_Shutdown

When invoked, this routine halts the Nucleus NAT module immediately, and de-allocates all resources previously allocated by the module. Any existing connections between the private and public network will be lost, and all portmap entries deleted. A successive call to [NAT\\_Initialize](#) will restart the Nucleus NAT module on the node.

### Usage

```
STATUS NAT_Shutdown (VOID);
```

### Arguments

None.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- Other operating system-specific error codes.

### Example

```
STATUS          status;  
...  
  
/* Initialize NAT. */  
...  
/* Un-initialize NAT. */  
status = NAT_Shutdown();  
  
...
```

### Related Topics

[NAT Functions](#)

[NAT Data Structures](#)

## Extended Protocol Support

Nucleus NAT will translate IP packets with the 8-bit protocol type of TCP, UDP and ICMP. If your NAT router must support other protocol types, you will have to modify the code to recognize each new protocol. NAT must be able to parse the packet, maintain connections established from either the private or external network, and remove the connections after a timeout period.

---

**i** **Tip:** To enable NAT in an application, include the `os/include/networking/nu_networking.h` file in your application code.

---

## Adding Support for Protocols that Contain Port Numbers

The following is an outline for adding support for protocols that contain a port number in the protocol header. You can use the code written for TCP as a guideline for the new protocol.

### Prerequisites

- None

### Procedure

1. Change macro definitions in the *os/include/networking/nat\_defs.h* file:
  - a. Declare a new macro, `NAT_MAX_XXXX_CONNS`, representing the number of simultaneous connections to be maintained of this protocol type, where `XXXX` is the name of the new protocol.
  - b. Add a new case to the logic to determine the value of `NAT_MAX_CONNS`. If `NAT_MAX_XXXX_CONNS` is greater than `NAT_MAX_TCP_CONNS` and `NAT_MAX_UDP_CONNS`, set `NAT_MAX_CONNS` to `NAT_MAX_XXXX_CONNS`.
  - c. Declare a new macro, `NAT_XXXX_TIMEOUT`, representing the number of seconds to wait before timing out a connection of this protocol type.
2. Change data structure definition in the file *os/include/networking/nat\_defs.h*:
  - a. Declare two new data structures; `NAT_XXXX_ENTRY` and `NAT_XXXX_TABLE`.
  - b. The `NAT_XXXX_TABLE` should contain a `NAT_XXXX_ENTRY` array of size `NAT_MAX_XXXX_CONNS` and an `INT32 nat_next_available_XXXX_index` representing the next entry in the array to be assigned to a new connection. Refer to `NAT_TCP_TABLE` as a guideline.
  - c. The `NAT_XXXX_ENTRY` should contain those parameters necessary to maintain connections of the new protocol type. Refer to `NAT_TCP_ENTRY` as a guideline.
3. Modify data structures within the *os/include/networking/nat\_defs.h* file:
  - a. Add a `NAT_XXXX_TABLE` pointer to the `NAT_TRANSLATION_TABLE` data structure.
  - b. Add the parameter `INT32 nat_next_avail_XXXX_port_index` to the data structure `NAT_PORT_LIST`.
  - c. Add any additional parameters to the `NAT_PACKET` data structure that must be parsed out of the packet for the new protocol.
4. Modify code within the *os/networking/nat/src/nat.c* file:

- a. NAT\_Initialize:
  - i. Allocate memory for the NAT\_Translation\_Table.nat\_xxxx\_table.
  - ii. Set NAT\_Translation\_Table.nat\_port\_list->nat\_next\_avail\_xxxx\_port\_index to 0.
  - iii. Initialize the members of the NAT\_XXXX\_TABLE.
  - iv. Set each instance of nat\_xxxx\_used\_flag in the port list to -1.
  - v. Set a new timer to timeout the entries in the NAT\_XXXX\_TABLE after NAT\_XXXX\_TIMEOUT seconds.
- b. NAT\_Portmap:

Add the protocol to fall through with the IPPROTO\_TCP and IPPROTO\_UDP cases in the switch statements within the NAT\_PORTMAP\_ADD and NAT\_PORTMAP\_DELETE cases. No additional code needs to be added.
- c. NAT\_Parse\_Packet:

Add a case for the protocol and add the appropriate parsing code. Refer to the parsing code in the IPPROTO\_TCP case as a guideline.
- d. NAT\_Find\_Translation\_Entry:

Add a case for the protocol and add the appropriate code to find a matching entry in the NAT\_XXXX\_TABLE based on the parameters parsed out of the packet. Refer to the IPPROTO\_TCP case as a guideline.
- e. NAT\_Add\_Translation\_Entry:

Add a case for the protocol and add the appropriate code to add an entry to the NAT\_XXXX\_TABLE. Note that this function also updates the nat\_next\_available\_xxxx\_index in the NAT\_XXXX\_TABLE. Refer to the IPPROTO\_TCP case as a guideline.
- f. NAT\_Delete\_Translation\_Entry:

Add a case for the protocol and add the appropriate code to delete an entry from the NAT\_XXXX\_TABLE.
- g. NAT\_Assign\_Port:

Add a case for the protocol and add the appropriate code to return the next available port and update the nat\_next\_avail\_xxxx\_port\_index in the port list.
- h. NAT\_Adjust\_Protocol\_Checksum:

Add a case for the protocol and add the appropriate code to adjust the checksum over the modified port number in the protocol header and the modified IP address in the

pseudoheader if the protocol contains a pseudoheader. Refer to the IPPROTO\_TCP case as a guideline.

i. NAT\_Translate:

- i. Add a new local variable of the type NAT\_XXXX\_ENTRY.
- ii. Within the code commented “Get a pointer to the appropriate entry based on the index”, add a case for the protocol and add the appropriate code to get a pointer to the entry. Also add a case for the protocol to the IPPROTO\_ICMP case. Refer to the IPPROTO\_TCP case as a guideline.
- iii. Within the code commented “Get the external port number”, add a case for the protocol and add the appropriate code to get the port number. Refer to the IPPROTO\_TCP case as a guideline.
- iv. Within the code commented “Get the internal IP address and port number”, add a case for the protocol and add the appropriate code to get the IP address and port number. Refer to the IPPROTO\_TCP case as a guideline.

j. NAT\_ICMP\_Parse\_Packet:

Add a case for the protocol to the switch statement to parse the required parameters from the embedded portion of an ICMP error packet.

5. Modify code within the *os/networking/nat/src/alg\_icmp.c* file:

a. ALG\_ICMP\_Translate

This function is the Application Level Gateway that modifies the embedded IP header and the first eight bytes of the payload of ICMP error packets. If you would like to be able to send and receive ICMP error packets of the new protocol type, you must modify this function in numerous places.

- i. Within the code commented “Get a pointer to the corresponding entry”, add a case for the protocol and add the appropriate code to get a pointer to the entry.
- ii. Within the code commented “Modify the IP address and port number of the embedded packet,” add a case for the protocol and add the appropriate code to translate the packet according to the parameters of the protocol header.
- iii. If changes were made to the first eight bytes of the embedded protocol header, and if the protocol checksum is contained within the first eight bytes of the embedded protocol header, within the code commented “Adjust the protocol checksum of the original message”, add a case for the protocol and the appropriate code to adjust the checksum over the changes.
- iv. If changes were made to the first eight bytes of the embedded protocol header, within the code commented, “Adjust the ICMP checksum over the protocol header changes”, add a case for the protocol and the appropriate code to adjust the ICMP checksum over the changes.

6. Modify code within *net/src/equeue.c* file:
  - a. `NU_EventsDispatcher`:

Add a case for `NAT_CLEANUP_XXXX` and add a function call to `NAT_Cleanup_XXXX`
7. Create new functions within *os/networking/nat/src/nat.c* file:
  - a. `NAT_Cleanup_XXXX`

This function traverses the list of entries in the `NAT_XXXX_TABLE` and removes those entries older than `NAT_XXXX_TIMEOUT`. Refer to the function `NAT_Cleanup_TCP` as a guideline.
  - b. `NAT_Delete_XXXX_Entry`

This function deletes an entry in the `NAT_XXXX_TABLE` with the specified index. Refer to the function `NAT_Delete_Translation_Entry` as a guideline.

## Related Topics

[Extended Protocol Support](#)

[Adding Support for Protocols Without Port Numbers](#)

## Adding Support for Protocols Without Port Numbers

The following is an outline for adding support for protocols that do not contain a port number in the protocol header. You can use the code written for ICMP as a guideline for the new protocol. Note that the sequence number and destination IP address in an ICMP packet are used to represent an ICMP connection. The private IP address, private port number, foreign IP address, and foreign port number in a TCP or UDP packet are used to represent a TCP or UDP connection. Your protocol must contain parameters, that when combined, can be used to represent a unique connection of the new protocol type.

## Prerequisites

- None

## Procedure

1. Modify macro definitions within the *os/include/networking/nat\_defs.h* file:
  - a. Declare a new macro, `NAT_MAX_XXXX_CONNS`, representing the number of simultaneous connections to be maintained of this protocol type, where `XXXX` is the name of the new protocol.
  - b. Add a new case to the logic to determine the value of `NAT_MAX_CONNS`. If `NAT_MAX_XXXX_CONNS` is greater than `NAT_MAX_TCP_CONNS` and

NAT\_MAX\_UDP\_CONNS, set NAT\_MAX\_CONNS to NAT\_MAX\_XXXX\_CONNS.

- c. Declare a new macro, NAT\_XXXX\_TIMEOUT, representing the number of seconds to wait before timing out a connection of this protocol type
2. Declare two new data structure definitions within the *os/include/networking/nat\_defs.h* file:
  - a. The NAT\_XXXX\_TABLE should contain a NAT\_XXXX\_ENTRY array of size NAT\_MAX\_XXXX\_CONNS and an INT32 nat\_next\_available\_XXXX\_index representing the next entry in the array to be assigned to a new connection. Refer to NAT\_ICMP\_TABLE as a guideline.
  - b. The NAT\_XXXX\_ENTRY should contain those parameters necessary to maintain connections of the new protocol type. Refer to NAT\_ICMP\_ENTRY as a guideline.
3. Modify data structures within the *os/include/networking/nat\_defs.h* file:
  - a. Add a NAT\_XXXX\_TABLE pointer to the NAT\_TRANSLATION\_TABLE data structure.
  - b. Add any additional parameters to the NAT\_PACKET data structure that must be parsed out of the packet for the new protocol.
  - c. Add any additional parameters to NAT\_PORTMAP\_SERVICE and NAT\_PORTMAP\_ENTRY that are necessary to represent connections of the protocol type.
4. Modify code within the *os/networking/nat/src/nat.c* file:
  - a. NAT\_Initialize
    - i. Allocate memory for the NAT\_Translation\_Table.nat\_XXXX\_table.
    - ii. Initialize the members of the NAT\_XXXX\_TABLE.
    - iii. Set a new timer to timeout the entries in the NAT\_XXXX\_TABLE after NAT\_XXXX\_TIMEOUT seconds.
  - b. NAT\_Portmap
    - i. Within the NAT\_PORTMAP\_ADD case, add a case for the new protocol. Add the appropriate code to check that the entry does not already exist, and add the new entry to the list.
    - ii. Within the NAT\_PORTMAP\_DELETE case, add a case for the new protocol. Add the appropriate code to find and delete the entry from the list.
  - c. NAT\_Find\_Portmap\_Entry
 

Within the code to find a matching entry, add a case comparing the necessary parameters to recognize an entry of the protocol type.

d. NAT\_Parse\_Packet

Add a case for the protocol and add the appropriate parsing code. Refer to the parsing code in the IPPROTO\_TCP case as a guideline.

e. NAT\_Adjust\_Protocol\_Checksum

If the new protocol contains a checksum, add a case for the protocol and add the appropriate code to adjust the checksum over the modified data in the protocol header and the modified IP address in the pseudoheader if the protocol contains a pseudoheader. Refer to the IPPROTO\_TCP case as a guideline.

f. NAT\_Translate

i. Add a new local variable of the type NAT\_XXXX\_ENTRY.

ii. Within the code commented, “Process the ICMP packet,” after the call to NAT\_Find\_Translation\_Entry, add a call to NAT\_Find\_XXXX\_Entry if an entry was not found in the previous call.

iii. Below the call to NAT\_Parse\_Packet and after the statement to check if the protocol type is ICMP, add an else if statement to check for the new protocol type. If the packet came from the external network, call NAT\_Find\_XXXX\_Entry. If the entry is found, assign the NAT\_XXXX\_ENTRY to the found entry. Otherwise, the packet came from the private network. Call NAT\_Add\_XXXX\_Entry.

iv. Within the code commented “Get a pointer to the appropriate entry based on the index”, add a case for the protocol and add the appropriate code to get a pointer to the entry. Also add a case for the protocol to the IPPROTO\_ICMP case. Refer to the IPPROTO\_TCP case as a guideline.

v. Within the code commented “Get the internal IP address and port number”, add a case for the protocol and add the appropriate code to get the IP address. Refer to the IPPROTO\_ICMP case as a guideline.

g. NAT\_ICMP\_Parse\_Packet

Add a case for the protocol to the switch statement to parse the required parameters from the embedded portion of an ICMP error packet.

5. Modify code within the *os/networking/nat/src/alg\_icmp.c* file:

a. ALG\_ICMP\_Translate

This function is the Application Level Gateway that modifies the embedded IP header and the first eight bytes of the payload of ICMP error packets. If you would like to be able to send and receive ICMP error packets of the new protocol type, you must modify this function in numerous places.

i. Within the code commented “Get a pointer to the corresponding entry”, add a case for the protocol and add the appropriate code to get a pointer to the entry.



- ii. Within the code commented “Modify the IP address and port number of the embedded packet”, add a case for the protocol and add the appropriate code to translate the entry according to the parameters of the protocol header.
  - iii. If changes were made to the first eight bytes of the embedded protocol header, and if the protocol checksum is contained within the first eight bytes of the embedded protocol header, within the code commented “Adjust the protocol checksum of the original message”, add a case for the protocol and the appropriate code to adjust the checksum over the changes.
  - iv. If changes were made to the first eight bytes of the embedded protocol header, within the code commented, “Adjust the ICMP checksum over the protocol header changes”, add a case for the protocol and the appropriate code to adjust the ICMP checksum.
6. Modify code within the *os/networking/net/src/equeue.c* file:
  - a. `NU_EventsDispatcher`

Add a case for `NAT_CLEANUP_XXXX` and add the function call to `NAT_Cleanup_XXXX`.
7. Add the following new functions to the *os/networking/nat/src/nat.c* file:
  - a. `NAT_Find_XXXX_Entry`

This function finds a matching entry in the `NAT_XXXX_TABLE` based on the parameters parsed out of the packet. Refer to the function `NAT_Find_ICMP_Entry` as a guideline.
  - b. `NAT_Add_XXXX_Entry`

This function adds an entry to the `NAT_XXXX_TABLE` based on the parameters parsed out of the packet. Note that this function also updates the `nat_next_available_XXXX_index` in the `NAT_XXXX_TABLE`. Refer to the function `NAT_Add_ICMP_Entry` as a guideline.
  - c. `NAT_Cleanup_XXXX`

This function traverses through the list of entries in the `NAT_XXXX_TABLE` and remove those entries older than `NAT_XXXX_TIMEOUT`. Refer to the function `NAT_Cleanup_ICMP` as a guideline.
  - d. `NAT_Delete_XXXX_Entry`

This function deletes an entry in the `NAT_XXXX_TABLE` with the specified index. Refer to the function `NAT_Delete_ICMP_Entry` as a guideline.

## Related Topics

[Extended Protocol Support](#)

[Adding Support for Protocols that Contain Port Numbers](#)

## Adding Additional Application Level Gateways (ALG)

An ALG is an extension to NAT, which modifies the payload of a packet aside from the IP and/or protocol headers.

Nucleus NET provides ALGs for FTP and ICMP. The FTP ALG is contained within the *os/networking/nat/src/alg\_ftp.c* file. It modifies the payload of PORT and PASV commands issued from the private network. Since this could change the size of the packet, a table exists for the FTP ALG to update sequence and acknowledge numbers of subsequent packets sent on this connection. The ICMP ALG is contained within the *os/networking/nat/src/alg\_icmp.c* file. It modifies the embedded IP header of ICMP error packets and the first eight bytes of the datagram as necessary. No additional table is required for the ICMP ALG.

### Prerequisites

- None

### Procedure

Depending on the application, use the following steps to add additional ALGs to NAT:

1. Your new ALG may or may not require additional macro definitions and/or data structures to complete its task. If necessary, add all new macro definitions and data structures to the following header file: *os/include/networking/alg\_defs.h*
2. Modify the following function code in the *os/networking/nat/src/alg.c* file:
  - i. `ALG_Modify_Payload`

Add a case to the switch statement for the protocol affected by the ALG. Call the function `ALG_XXXX_Translate`.

- a. This function performs the necessary modification to the packet

### Related Topics

[Extended Protocol Support](#)



















## DHCP Overview

The Dynamic Host Configuration Protocol (DHCP) offers a means for providing Internet hosts with configuration parameters. DHCP is built on a server/client model where the server is responsible for the allocation of network addresses and providing configuration parameters to dynamically configured clients. The server must be able to manage a pool of IPv4 network addresses and allocate these addresses to requesting clients on a per case basis. One advantage of using DHCP is that a large client base can be served with fewer network addresses if each client is not in need of a full time network connection. Instead of permanently assigning a network address to a client, the address can be allocated to a client for a period of time on an as-needed basis. If the client is no longer in use once the defined period of time has passed, the network address can be allocated to another client. However, if the client is still in use, it can request an extension for use of the address. This is the concept behind the use of a lease time for all addresses allocated by the server.

The server not only provides a network address to the client, but can also provide numerous network configuration parameters. This is another advantage that DHCP provides over manually configured clients. Parameters such as router addresses, DNS server addresses, and network domain name can be changed for every client on a network that uses DHCP to obtain its network parameters rather than having to manually change each machine.

In certain situations, it is desired that a specific host always obtain the same IPv4 address, such as a web server. The DHCP server provides a method for setting aside certain IPv4 addresses so that they may only be issued to one particular client. The IPv4 address becomes tied to the MAC address of the client that should be granted the IPv4 address.

With the wide array of configurable options that may be provided to clients, it may be desired to present different values for the same options to certain clients. The DHCP server provides a mechanism for setting certain options to be used by clients that obtain certain IPv4 addresses.

The requirements for a DHCP server are outlined in RFC 2131.

## Theory of Operation

The process in which a DHCP server provides configuration parameters to a client involves the following four parts:

- **DISCOVER**
- **OFFER**

- [REQUEST](#)
- [ACK](#)

## DISCOVER

The client that desires to receive configuration parameters from a DHCP server will send out a DISCOVER message on the network to which it is physically connected. The message will be broadcast over the network bound for the DHCP server port of any device connected to the network. The DISCOVER message will contain certain information about the client, such as its hardware address.

## OFFER

Once a server receives a DISCOVER message, it must determine if it should offer a network address to the requesting client. Once it is determined that the requesting client should be offered an address, the server must choose an address from its available pool of addresses. After the address has been chosen, an OFFER message will be broadcast over the network from which the message originated.

## REQUEST

If a client decides to accept an offer from a DHCP server, the client will send out a REQUEST message requesting the offered IPv4 address. This message will contain the server network address of the server that offered the address that is being accepted. The client may also request certain configuration parameters at this time that were not requested in the DISCOVER message. This message is also broadcast over the network to which the client is physically connected.

## ACK

Once the server receives a REQUEST message from a client, the server can mark the network address that was offered to the client as in use and should not offer it to another client until the client notifies that it has released it. The server will also go through the configuration parameters requested by the client and provide these parameters, if supported by the server, to the client in an ACK message.

Once the client receives the ACK message from the server, it is free to use the network address for the defined lease time. At any time before the lease time has elapsed, the client could send a REQUEST message to renew its lease of the network address. After the lease time has expired and the client has not successfully renewed the lease, the server must mark the address as available for lease by any client that may request an address.

## DHCP Server Requirements

The Nucleus DHCP server requires a memory file system to keep a record of any and all leases that are in use. A target with disk-support will be able to fully utilize the Nucleus FILE support through being able to retrieve a record of the leases after a complete restart of the server. An in-memory file system may be used, but will not provide the support of recovering the record of leases through a server restart.

## Boot-up Configuration of DHCP Server

A DHCP server needs to be configured with network configuration options and IP addresses for clients. These configuration parameters can be presented to the DHCP server in two ways:

- Previously saved configurations
- API functions

During initialization, the DHCP server will search for previously saved configuration and binding files on the file system. If these files are not in place, the server will continue with its initialization as normal and will acquire its parameters during run-level initialization.

The Nucleus DHCP server is initialized with most commonly used parameters that can be changed through metadata options. These options include the IP address range, Router Address, Domain Name Server Addresses, Network Interface Name, DHCP filenames and drive path, lease time and rebind time and so on. The metadata options simplify the environment. You can also use the available APIs to get the complete control of the server. For more information on the use of the API functions, refer to the [DHCP Server API Function Reference](#) section of this chapter. The following are the configuration parameters and their default values.

### default\_drive

Set this to drive number to be used for storing records of any and all leases that are in use (0 for 'a:', 1 for 'b:' and so on). The default value is 0.

### directory\_path

Directory path and name for DHCP server directory that stores DHCP related files. Keep the drive letter according to the default\_drive option. The default path is "a:\\\\dhcps".

### interface\_name

Name of the network interface over which DHCP server will run. The default interface is "eth0".

### **subnet\_mask**

Subnet mask for the network. This is used to determine if a specified IP range is valid before adding it to a control block. The default subnet mask is "255.255.255.0".

### **include\_router**

Enable for DHCP Router option. Setting this to false will exclude router functionality from the build for the DHCP server. The default value is true.

### **router1\_address**

Address of the router on the network. The default address is "200.100.200.5".

### **include\_dns\_server**

Enable for DHCP DNS Server option. Setting this to false will exclude the domain name server functionality from the build for the DHCP server. The default value is true.

### **domain\_server1\_address**

First DNS Server Address. Set this address to the first Domain Name Server address on your Network. The default address is "200.100.200.2".

### **domain\_server2\_address**

Second DNS Server Address. Set this address to the second Domain Name Server address on your Network. The default address is "200.100.200.3".

### **broadcast\_address**

Broadcast address on the network. A message sent to this broadcast address is received by all network-attached hosts. The default address is "200.100.200.255".

### **subnet\_address**

Subnet address for this network. This establishes the correct control block from which to issue parameters to requesting clients. The default address is "200.100.200.0".

### **ip\_range\_low**

Starting address of the IP range from which IP addresses will be assigned to incoming clients. The default starting IP address is "200.100.200.10".

### **ip\_range\_high**

Ending address of the IP range from which IP addresses will be assigned to incoming clients. The default ending IP address is "200.100.200.20".

## **include\_domain\_name**

Enable for DHCP domain name option. Setting this to false will exclude domain name functionality of the DHCP server from the build. The default value is true.

## **domain\_name**

Domain name. Set this to your domain name. The default is "yourdomain.com".

## **lease\_time**

Lease time in seconds. Sets the amount of time that the clients can hold a binding without renewing/rebinding. The default is 300 seconds.

## **renewal\_t1\_time**

Renewal time in seconds. After this portion of the lease time has expired, the client will attempt to renew its lease to keep using its IP address. The default is 150 seconds.

## **rebind\_t2\_time**

Rebind time in seconds. If the client is unable to renew its lease within this time, the lease will expire. The client then needs to obtain a new lease. The default is 225 seconds.

## **offered\_wait\_time**

Wait time for an offered IP address. The default is 150 seconds.

# **User Configurable Defines**

The user-configurable region for the Nucleus DHCP Server is located in *os/include/networking/dhcps\_cfg.h*. The macros in this can be configured by the user for the values to be used by the server.

## **DHCP\_MAX\_ENTRY\_NAME\_LENGTH**

This macro is the maximum number of characters that can be used for the entry name of a configuration. The default value is 8.

## **DHCP\_LEASE\_TIMER\_SLEEP**

This macro is the interval between savings of the binding structures to a file. This saving of the binding structures needs to take place at regular intervals in case of a server restart. The default value is 60 seconds.

## DEFAULT\_DECLINED\_LEASE\_LENGTH

If an offered binding is declined, the IP address associated with that binding is in use by another client. Therefore, this binding will be marked as `IN_USE` for this period of time so that it will not be immediately offered to another client. The default value is 360 seconds.

## DHCP\_FILE\_READ\_SIZE

This macro is the value for the block of memory that will be read from a file at initialization. If memory limitations are a concern, this macro can be set to a lower value. However, this may lengthen the initialization process, due to the need for multiple file reads. The default value is 512.

## DHCP\_MAX\_FILE\_NAME\_SIZE

This macro is the maximum length of the file names for the configuration and binding files. This includes the file extension as well. The default value is 20.

## DHCP\_MAX\_STRING\_LENGTH

This macro is the maximum length for a character string. If a character string is longer than this value, it will be truncated to the `MAX_STRING_LEN`. The default value is 260.

## DHCP\_MAX\_DOMAIN\_NAME\_SIZE

This macro is the maximum length of the domain name. The default value is 16.

## DHCP\_MAX\_CLIENT\_ID\_LENGTH

This macro is the maximum length for a client ID that the server will encounter. The client ID is a unique combination of characters that the server can use to distinguish between different clients. The default value is 16.

## DHCP\_MAX\_DNS\_SERVERS

This macro is the maximum number of DNS servers that can be added to an individual configuration control block. The default value is 3.

## DHCP\_MAX\_ROUTERS

This macro is the maximum number of routers that can be added to an individual configuration control block. The default value is 3.

## DHCP\_OPTIONS\_MEMORY\_BLOCK\_SIZE

This macro is the size of the memory block that will be used to store option values that are specific to a certain binding. The default value is 200.



## DHCP Server User Control

You can disable the initial configuration done during the run-level initialization. You can do this by first using the [DHCP Server Reset](#) function, then by using the available APIs to configure the DHCP server according to your requirements.

You first must create a configuration control block for holding the DHCP server parameters. The API function call [DHCP Create Config Entry](#) will create the configuration control block and prepare it to receive option parameters. A name is associated with each individual control block. A control block named "GLOBAL" will have properties that apply to all control blocks. Additional control blocks to specify different sets of parameters may be added. The entries present in a configuration control block are as follows:

## DHCP Configuration Control Block

The DHCP configuration control block holds the configuration data that will be presented to a requesting client. Each entry of the control block has a corresponding API function to add or modify the entries.

Table [5-1 DHCP Configuration Control Block](#), is an overview of each entry in the structure.

**Table 5-1. DHCP Configuration Control Block**

Structure Entry	Description
dhcp_config_next	Pointer to the next control block of the link list.
dhcp_config_previous	Pointer to the previous control block of the link list.
server_ip_addr	The network IPv4 address of the DHCP server.
subnet_addr	Subnet address of the binding IPv4 address associated with the configuration control block.
subnet_mask_addr	Subnet mask to be offered to client.
broadcast_addr	Broadcast address to be offered to client.
ip_binding_list	Binding IP link list.
ip_offer_list	Offered IP link list.
dns_domain_name	DNS domain name to be presented to the requesting clients.
dns_domain_name_length	Length of the domain name.
dns_server	Array of DNS server IPv4 addresses to be presented to the requesting clients.
router	Array of router IPv4 addresses to be presented to the requesting clients.

**Table 5-1. DHCP Configuration Control Block (cont.)**

Structure Entry	Description
device_name	Name of network device interface that is associated with the control block.
config_entry_name	Character string that is used to identify each control block. "GLOBAL" is the only restricted name.
default_ip_ttl	IPv4 time to live to be presented to the requesting clients.
default_tcp_ttl	TCP time to live to be presented to the requesting clients.
flags	Status flags of the control block: <ul style="list-style-type: none"> <li>• GLOBAL_CONFIGURATION</li> <li>• TCP_KEEPALIVE_GARBAGE</li> <li>• IP_FORWARDING_ENABLED</li> <li>• ETHERNET_IEEE 802_ENCAPSULATION</li> <li>• TRAILER_ENCAPSULATION</li> <li>• CONFIGURATION_ENABLED</li> </ul>
dhcp_renew_t1	Period of time (in seconds) that should elapse before the client needs to attempt to renew its lease.
dhcp_rebind_t2	Period of time (in seconds) that should elapse before the client needs to attempt to rebind to another lease.
default_lease_length	Default lease length (in seconds) that will be offered to a client that does not request a certain lease period.
max_lease_length	Maximum lease time (in seconds) that can be offered to a client.
offered_wait_time	Default wait time (in seconds) that should elapse before a binding that has been offered to a client can be offered to another client.
tcp_keepalive_interval	TCP keepalive interval (in seconds) that will be presented to requesting clients.
arp_cache_to	ARP cache timeout value (in seconds) that will be presented to requesting clients.
options_buff	Special options buffer link list.

You may not add and/or modify each entry in Table 5-1, but the options in Table 5-2 must be added to either the global control block or each individual control block.

**Table 5-2. Required Control Block Options**

Option	Description
Subnet Address	Establishes the control block from which to issue parameters to requesting clients.
Subnet Mask	Determines if a specified IP range is valid before adding it to a control block.
Default Lease Time	Sets the amount of time that a client can hold a binding without renewing/rebinding.
Renewal Time	Sets the renewal time that will be provided to the requesting clients.
Rebinding Time	Sets the rebinding time that will be provided to the requesting clients.

## Adding Supplementary Options

The DHCP protocol supports numerous options. Some options are not supported by this release of the Nucleus DHCP Server. To implement an unsupported option, you will need to perform several steps to ensure the option will perform correctly. The examples listed for each step add the TFTP server name option.

1. Add the new option to the `DHCPS_CONFIGURATION` structure. This structure contains the assigned values for each of the supported options of the server and must have a new entry added for a new option.

### Note



If the new option is only one byte in length, such as options that enable or disable a parameter, the option can be supported by a bit in the status flag rather than allocating a new variable.

Example:

```
CHAR tftp_server_name[16];
```

2. You need to add a macro to the `dhcpsdb.h` file that contains the option tag value. If the desired option to be added is not present, it can be added.

Example:

```
#define TFTP_SERVER_NAME 66
```

3. You must add a function in the `dhcpsdb.h` file to handle the data type of the option. The present functions are: `DHCPSDB_Process_IP`, `DHCPSDB_Process_Host_Long`, `DHCPSDB_Process_Host_Short`, and `DHCPSDB_Process_String`. Also, you must add

a case statement to the handling function for the new option. The case statement must handle the verifying and copying of the option data into the configuration control block.

Example:

```
DHCP_SDB_Process_Stringcase TFTP_SERVER_NAME:
status = DHCP_SDB_Get_String(symbol, str_ptr);
if (status == NU_SUCCESS)
{
    strcpy(config->tftp_server_name, str_ptr);
    break;
}
```

4. You must add an entry to the function `DHCP_SDB_Supported_Options_List` so that a four character tag and the data type handling function can be associated with the option.

Example:

```
if (strcmp(opcode, "tftp") == 0){
    optionmap_ptr->opcode = 'tftp';
    optionmap_ptr->option_name = TFTP_SERVER_NAME;
    option_map_ptr->func = DHCP_SDB_Process_String;
    ret_status = NU_SUCCESS;
}
```

5. To provide the option data to requesting clients, you must add a case statement for the new option to the function `DHCP_Get_Option_From_Configuration`.

Example:

```
case TFTP_SERVER_NAME:
*option_buff++ = TFTP_SERVER_NAME;
*option_buff++ = (UINT8)strlen(config_cb->tftp_server_name);
strcpy((CHAR*)option_buff, config_cb->tftp_server_name);
option_buff += (UINT8)strlen(config_cb->tftp_server_name);
option_data_lenth +=
    (UINT8)(2 + strlen(config_cb->tftp_server_name))
break;
```

6. To save the option to a file, you must add a procedure to the `DHCP_Save_Config_Struct_To_File` function. This procedure places the option tag and the option data into a buffer so that it can be written out to a file.

Example:

```
strcpy(buffer_pointer, ":tftp=");
buffer_pointer += 6;
bytes_written += 6;
data_len = strlen(config_cb->tftp_server_name);
memcpy(buffer_pointer, config_cb->tftp_server_name, data_len);
buffer_pointer += data_len;
bytes_written += data_len;
```

7. Create APIs for adding, deleting, and retrieving option data. These functions may be written many different ways. You may use the current API functions as a guideline for developing your own functions.

## DHCP Data Structures

### id\_struct

id\_struct is a 32-bit structure containing 4-digit IP address.

The definition is as follows:

```
struct id_struct
{
    UINT8 is_ip_addrs[MAX_ADDRESS_SIZE]; /* IP address number */
};
```

Where for IPv4 MAX\_ADDRESS\_SIZE is equal to 4.

### Related Topics

<a href="#">DHCP Data Structures</a>	<a href="#">DHCPS_Add_Broadcast_Address</a>
<a href="#">DHCPS_Add_DNS_Server</a>	<a href="#">DHCPS_Add_IP_Range</a>
<a href="#">DHCPS_Add_Router</a>	<a href="#">DHCPS_Delete_Broadcast_Address</a>
<a href="#">DHCPS_Delete_DNS_Server</a>	<a href="#">DHCPS_Delete_Domain_Name</a>
<a href="#">DHCPS_Delete_IP_Range</a>	<a href="#">DHCPS_Delete_Router</a>
<a href="#">DHCPS_Get_ARP_Cache_Timeout</a>	<a href="#">DHCPS_Get_Broadcast_Address</a>
<a href="#">DHCPS_Get_Default_Lease_Time</a>	<a href="#">DHCPS_Get_DNS_Servers</a>
<a href="#">DHCPS_Get_Domain_Name</a>	<a href="#">DHCPS_Get_IP_TTL</a>
<a href="#">DHCPS_Get_Offered_Wait_Time</a>	<a href="#">DHCPS_Get_Rebind_Time</a>
<a href="#">DHCPS_Get_Renewal_Time</a>	<a href="#">DHCPS_Get_Router</a>
<a href="#">DHCPS_Get_Subnet_Address</a>	<a href="#">DHCPS_Get_Subnet_Mask</a>
<a href="#">DHCPS_Get_TCP_Keepalive_Interval</a>	<a href="#">DHCPS_Get_TCP_TTL</a>
<a href="#">DHCPS_Set_ARP_Cache_Timeout</a>	<a href="#">DHCPS_Set_Default_Lease_Time</a>
<a href="#">DHCPS_Set_IP_TTL</a>	<a href="#">DHCPS_Set_Offered_Wait_Time</a>
<a href="#">DHCPS_Set_Rebind_Time</a>	<a href="#">DHCPS_Set_Renewal_Time</a>
<a href="#">DHCPS_Set_Subnet_Address</a>	<a href="#">DHCPS_Set_Subnet_Mask</a>
<a href="#">DHCPS_Set_TCP_Keepalive_Interval</a>	<a href="#">DHCPS_Set_TCP_TTL</a>

## DHCPS\_CLIENT\_ID

DHCPS\_CLIENT\_ID is the typedef name for the \_DHCPS\_CLIENT\_ID data structure, and it is defined as follows. This structure is used to store the client identifier option fields of the DHCP message.

```
typedef struct _DHCPS_CLIENT_ID
{
    CHAR            idtype;
    UINT8           idlen,
                   id[DHCPS_MAX_CLIENT_ID_LENGTH],
                   pad1[2];
}DHCPS_CLIENT_ID;
```

Where the members of the structure are defined as in [Table 5-3](#).

**Table 5-3. DHCP\_CLIENT\_ID**

Member	Description
idtype	Indicates whether the client identifier is a hardware address (idtype == 1) or other than a hardware address.
idlen	Length of the client identifier.
id[DHCPS_MAX_CLIENT_ID_LENGTH]	Client identifier.
pad1[2]	Reserved.

### Related Topics

[DHCP Data Structures](#)

[DHCPS\\_Add\\_IP\\_Range](#)

[DHCPS\\_Delete\\_IP\\_Range](#)

## DHCP Server API Function Reference

The Nucleus DHCP Server utilizes API calls to add and remove option data. Certain API functions can also retrieve data from the server. The following is a listing of each of the API functions of the Nucleus DHCP Server:

- [DHCPS\\_Server\\_Reset](#)
- [DHCPS\\_Add\\_Broadcast\\_Address](#)
- [DHCPS\\_Add\\_DNS\\_Server](#)
- [DHCPS\\_Add\\_Domain Name](#)
- [DHCPS\\_Add\\_IP\\_Range](#)
- [DHCPS\\_Add\\_Router](#)

- [DHCPS\\_Create\\_Config\\_Entry](#)
- [DHCPS\\_Delete\\_Broadcast\\_Address](#)
- [DHCPS\\_Delete\\_Config\\_Entry](#)
- [DHCPS\\_Delete\\_DNS\\_Server](#)
- [DHCPS\\_Delete\\_Domain\\_Name](#)
- [DHCPS\\_Delete\\_IP\\_Range](#)
- [DHCPS\\_Delete\\_Router](#)
- [DHCPS\\_Disable\\_Configuration](#)
- [DHCPS\\_Disable\\_Ethernet\\_IEEE\\_Encapsulation](#)
- [DHCPS\\_Disable\\_IP\\_Forwarding](#)
- [DHCPS\\_Disable\\_TCP\\_Keepalive\\_Garbage](#)
- [DHCPS\\_Disable\\_Trailer\\_Encapsulation](#)
- [DHCPS\\_Enable\\_Configuration](#)
- [DHCPS\\_Enable\\_Ethernet\\_IEEE\\_Encapsulation](#)
- [DHCPS\\_Enable\\_IP\\_Forwarding](#)
- [DHCPS\\_Enable\\_TCP\\_Keepalive\\_Garbage](#)
- [DHCPS\\_Enable\\_Trailer\\_Encapsulation](#)
- [DHCPS\\_Get\\_ARP\\_Cache\\_Timeout](#)
- [DHCPS\\_Get\\_Broadcast\\_Address](#)
- [DHCPS\\_Get\\_Config\\_Entry](#)
- [DHCPS\\_Get\\_Default\\_Lease\\_Time](#)
- [DHCPS\\_Get\\_DNS\\_Servers](#)
- [DHCPS\\_Get\\_Domain\\_Name](#)
- [DHCPS\\_Get\\_IP\\_Range](#)
- [DHCPS\\_Get\\_IP\\_TTL](#)
- [DHCPS\\_Get\\_Offered\\_Wait\\_Time](#)
- [DHCPS\\_Get\\_Rebind\\_Time](#)
- [DHCPS\\_Get\\_Renewal\\_Time](#)
- [DHCPS\\_Get\\_Router](#)

- [DHCPS\\_Get\\_Subnet\\_Address](#)
- [DHCPS\\_Get\\_Subnet\\_Mask](#)
- [DHCPS\\_Get\\_TCP\\_Keepalive\\_Interval](#)
- [DHCPS\\_Get\\_TCP\\_TTL](#)
- [DHCPS\\_Set\\_ARP\\_Cache\\_Timeout](#)
- [DHCPS\\_Set\\_Default\\_Lease\\_Time](#)
- [DHCPS\\_Set\\_IP\\_TTL](#)
- [DHCPS\\_Set\\_Offered\\_Wait\\_Time](#)
- [DHCPS\\_Set\\_Rebind\\_Time](#)
- [DHCPS\\_Set\\_Renewal\\_Time](#)
- [DHCPS\\_Set\\_Subnet\\_Address](#)
- [DHCPS\\_Set\\_Subnet\\_Mask](#)
- [DHCPS\\_Set\\_TCP\\_Keepalive\\_Interval](#)
- [DHCPS\\_Set\\_TCP\\_TTL](#)
- [DHCPS\\_Shutdown\\_Server](#)



## DHCPD\_Server\_Reset

This function disables and deletes all DHCP server configurations that were done during run-level initialization.

### Usage

```
STATUS DHCPD_Server_Reset ();
```

### Arguments

- none

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; all DHCP server configuration done during initialization were disabled.
- **NU\_INVALID\_PARM**  
The specified configuration structure was not found in the configuration link list.
- **DHCPDERR\_PARAMETER\_NOT\_REMOVED**  
An error occurred while removing the control block from the link list. The control block is still on the link list and must be still deleted.
- **DHCPDERR\_NOT\_INITIALIZED**  
The DHCP server is in the process of initialization. The user must wait for this function to return success.

### Description

The function searches and deletes configuration entries named GLOBAL and SUBNET1 created during run-level initialization. You must call this function before controlling the DHCP server through APIs.

### Example

```
#include "networking/nu_networking.h"
#include "dhcpd.h"

STATUS status;

/* Reset the server so that the initial configurations done by Nucleus are
   reverted and user can control the DHCP server through APIs. */

while ((status = DHCPD_Server_Reset()) != NU_SUCCESS)
{
    NU_Sleep(100);
}
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Shutdown\\_Server](#)

## DHCPs\_Add\_Broadcast\_Address

This function adds a broadcast address to the specified configuration control block pointed to by `config_cb`.

### Usage

```
STATUS DHCPs_Add_Broadcast_Address (DHCPs_CONFIG_CB    *config_cb,
                                   struct id_struct     *client_ip_addr,
                                   struct id_struct     *broadcast_addr);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the broadcast address will be placed.
- `client_ip_addr`  
If the desired broadcast address is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `broadcast_addr`  
Pointer to the structure that contains the broadcast address. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; the broadcast address has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
One of the passed in parameters is invalid. The broadcast address was not added.

### Example

```
STATUS          status;
DHCPs_CONFIG_CB *config_cb;
struct id_struct broadcast_addr;
CHAR            DHCP_Broadcast_Addr[] = {200,100,200,255};

/* Reset the server so that the initial configurations done by Nucleus are
   reverted and user can control the DHCP server through APIs. */

status = DHCPs_Server_Reset();

if (status == NU_SUCCESS)
{
    /* Create a global configuration global control block.*/
    if (DHCPs_Create_Config_Entry(&config_cb,"GLOBAL",0)==NU_SUCCESS)
    {
        /* Copy the broadcast address into a structure. */
    }
}
```

```
        memcpy(&broadcast_addr, DHCP_Broadcast_Addr, IP_ADDR_LEN);

        /* Add the broadcast address to the global control
           block. */
        status = DHCPS_Add_Broadcast_Address(config_cb, NU_NULL,
                                              &broadcast_addr);
    }
}
```

## Related Topics

[DHCP Server API Function Reference](#)[DHCPS\\_Delete\\_Broadcast\\_Address](#)[DHCPS\\_Get\\_Broadcast\\_Address](#)

## DHCPs\_Add\_DNS\_Server

This function adds the DNS server IPv4 address to the array of DNS server IPv4 addresses contained in the configuration control block pointed to by `config_cb`. The size of the array is specified by `MAX_DNS_SERVERS`, which can be found in the file *dhcpscfg.h*.

### Usage

```
STATUS DHCPs_Add_DNS_Server (DHCPs_CB      *config_cb,  
                             struct id_struct *client_ip_addr,  
                             struct id_struct *dns_ip_address);
```

### Arguments

- `config_cb`  
Pointer to the configuration block where the DNS server will be placed.
- `client_ip_addr`  
If this is a DNS server IPv4 address that is be specific to one binding address, then set the argument `client_ip_addr` to the address of the desired binding. Otherwise, set this to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `dns_ip_address`  
Pointer to the structure containing the IPv4 address of the DNS server. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the DNS server has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
An error occurred during the adding of the DNS server to the configuration structure and the server has not been added.
- `DHCPsERR_ARRAY_ENTRIES_FULL`  
The array that stores the DNS server IPv4 addresses is full. The specified IPv4 address was not added.
- `DHCPsERR_OPTION_ALREADY_PRESENT`  
The specified DNS server IPv4 address was already in the array of IPv4 addresses. No further action was taken.

### Example

```
STATUS      status;  
struct id_struct dns_ip_address;  
DHCPs_CONFIG_CB *config_cb;  
CHAR      Domain_Server_Addr[] = {200,100,200,2};
```

```
/* Assumes that the server reset has been done and a global config entry
was created successfully in config_cb */
.....

/* Copy the DNS address into a structure. */
memcpy(&dns_ip_address, Domain_Server_Addr, IP_ADDR_LEN);

/* Add the broadcast address to the global control
block. */
status = DHCPS_Add_DNS_Server(config_cb, NU_NULL, &dns_ip_address);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Delete\\_DNS\\_Server](#)

[DHCPS\\_Get\\_DNS\\_Servers](#)

## DHCPD\_Add\_Domain Name

This function adds the domain name to the specified configuration control block. You can set the maximum domain name length using the macro `MAX_DOMAIN_NAME_LENGTH`, which is found in the *dhcps\_cfg.h*.

### Usage

```
STATUS DHCPD_Add_Domain_Name (DHCPD_CB      *config_cb,  
                             struct id_struct *client_ip_addr,  
                             CHAR            *domain_name);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the DNS domain name will be placed.
- `client_ip_addr`  
If the domain name is meant for a specific binding, then set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `domain_name`  
Pointer to the buffer that contains the domain name.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; the DNS domain name has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
Either the configuration control block pointer or the pointer to the domain name was `NU_NULL`. The domain name was not added.
- `DHCPDERR_OPTION_ALREADY_PRESENT`  
A domain name was already present. The current domain name must be removed before a new domain name may be added. The domain name may be removed using the function [DHCPD\\_Delete\\_Domain\\_Name](#).

### Example

```
STATUS      status;  
CHAR        *domain_name = "mydomain.com";  
DHCPD_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a global config entry  
was created successfully in config_cb */  
.....  
  
/* Add the domain name to the global control block*/  
status = DHCPD_Add_Domain_Name(config_cb,NU_NULL, domain_name);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Delete\\_Domain\\_Name](#)

[DHCPS\\_Get\\_Domain\\_Name](#)



## DHCPAdd\_IP\_Range

This function allocates binding structures for each of the IPv4 addresses in the range between the `ip_low` and `ip_high`. It also allocates the memory for the binding structures and links it to the binding structure link list.

---

**Note**

A valid subnet mask must be in place in either the specific or global configuration control block.

---

### Usage

```
STATUS DHCPAdd_IP_Range (DHCP_CB          *config_cb,  
                        DHCP_CLIENT_ID *client_id,  
                        struct id_struct *ip_low,  
                        struct id_struct *ip_high);
```

### Arguments

- `config_b`  
Pointer to the configuration control block associated with each binding.
- `client_id`  
Pointer to the structure that contains the client identifier of the incoming client in case of static binding only. Set to `NU_NULL` otherwise. Data structure [DHCP\\_CLIENT\\_ID](#) is defined in the [Related Topics](#) section of this chapter.
- `ip_low`  
Pointer to the structure that contains the lowest numeric IPv4 address of the range. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- `ip_high`  
Pointer to the structure that contains the highest numeric IPv4 address of the range. If this is a static binding, this argument should be set to `NU_NULL`. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; added each of the IPV4 addresses in the range.
- `DHCPERR_RANGE_SUBNET_INVALID`  
All of the IPv4 addresses in the specified range cannot be rectified to be on the same subnet with the available subnet masks. None of the IP bindings were added to the list.
- `DHCPERR_BINDING_NOT_ADDED`  
An error occurred while attempting to add a binding onto the link list. Part of the range may have been added to the link list. If this error is returned, it is best to perform a [DHCPGet\\_IP\\_Range](#) to determine which bindings, if any, were added.

## Example

```
STATUS          status;
DHCPS_CONFIG_CB *config_cb;
CHAR            IP_Range_Low[] = {200, 100, 200, 30};
CHAR            IP_Range_High[] = {200, 100, 200, 40};
struct id_struct ip_low, ip_high;

/* Assumes that the server reset has been done and a config entry was
created successfully in config_cb */
.....

/* Copy the IP range high and low into id_structs. */
memcpy(&ip_low, IP_Range_Low, IP_ADDR_LEN);
memcpy(&ip_high, IP_Range_High, IP_ADDR_LEN);

/* Add IP range to the control block. */
status = DHCPS_Add_IP_Range(config_cb, NU_NULL, &ip_low, &ip_high);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Delete\\_IP\\_Range](#)

[DHCPS\\_Get\\_IP\\_Range](#)

## DHCPs\_Add\_Router

This function adds the router IPv4 address to the array of router IPv4 addresses contained in the configuration control block pointed to by `config_cb`. The size of the array is specified by `MAX_ROUTERS`, which can be found in the `dhcps_cfg.h` file.

### Usage

```
STATUS DHCPs_Add_Router (DHCPs_CB      *config_cb,  
                        struct id_struct *client_ip_addr,  
                        struct id_struct *router_address);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the router will be placed.
- `client_ip_addr`  
If this router is meant for only one particular IPv4 address, set this argument to that address. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `router_address`  
Pointer to the structure containing the IPv4 address of the router. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; added router to the specified configuration structure.
- `NU_INVALID_PARM`  
An error occurred while adding of the router to the configuration structure and the server has not been added.
- `DHCPsERR_ARRAY_ENTRIES_FULL`  
The array that stores the router IPv4 addresses is full. The specified IPv4 address was not added.
- `DHCPsERR_OPTION_ALREADY_PRESENT`  
The specified router IPv4 address was already in the array of IPv4 addresses. No further action was taken.

### Example

```
STATUS      status;  
struct id_struct router_address;  
DHCPs_CONFIG_CB *config_cb;  
CHAR Router_Addr[] = {200,100,200,1};
```

```
/* Assumes that the server reset has been done and a global config entry
was created successfully in config_cb */
.....

/* Copy the router address into a structure. */
memcpy(&router_address, Router_Addr, IP_ADDR_LEN);

/* Add the router address to the global control block. */
status = DHCPS_Add_Router(config_cb, NU_NULL, &router_address);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Delete\\_Router](#)

[DHCPS\\_Get\\_Router](#)

## DHCPCreateConfigEntry

This function creates a new configuration control block to house the configuration properties of the Nucleus DHCP Server. The function allocates the memory and links the memory to the control block link list.

### Usage

```
STATUS DHCPCreateConfigEntry (DHCP_CONFIG_CB **config_cb,
                             CHAR             *config_name,
                             CHAR             *interface_name);
```

### Arguments

- **config\_cb**  
Pointer to a pointer to the configuration control block being created. The config\_cb pointer is set to the location of the allocated memory.
- **config\_name**  
Pointer to the character string used as the name of the control block. The config\_name associates bindings with the control block. The config\_name of "GLOBAL" is reserved for the control block containing the parameters that apply to all control blocks.
- **interface\_name**  
Pointer to the network device interface name. The interface\_name associates a control block with a certain network interface.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the specified configuration entry has been added.
- **DHCPERR\_CONFIGURATION\_PRESENT**  
A control block with the same config\_name is already on the link list. The control block will not be added.
- **NU\_INVALID\_PARM**  
No interface\_name is specified and the control block is not intended to be global. The control block will not be added.

### Example

```
STATUS          status;
DHCP_CONFIG_CB *config_cb;

/*Create a global config entry */
status = DHCPCreateConfigEntry(&config_cb, "GLOBAL", 0);

/* OR */

/*Create a specific config entry */
```

```
status = DHCPS_Create_Config_Entry(&config_cb, "SUBNET1", "eth0");
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Delete\\_Config\\_Entry](#)

[DHCPS\\_Get\\_Config\\_Entry](#)

## DHCP Delete Broadcast Address

This function removes the broadcast address from the specified configuration control block.

### Usage

```
STATUS DHCP Delete Broadcast Address (DHCP_CONFIG_CB *config_cb,  
                                     struct id_struct *client_ip_addr,  
                                     struct id_struct *broadcast_addr);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the broadcast address will be removed.
- **client\_ip\_addr**  
If the desired broadcast address is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- **broadcast\_addr**  
Pointer to the structure containing the broadcast address. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter. [id\\_struct](#)

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the broadcast address has been removed from the specified configuration structure.
- **NU\_INVALID\_PARMS**  
Either the config\_cb or broadcast\_addr pointers were set to NU\_NULL.
- **DHCPERR\_PARAMETER\_NOT\_FOUND**  
The specified broadcast address was not found in specified configuration control block.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
struct id_struct broadcast_addr;  
CHAR            DHCP_Broadcast_Addr[] = {200,100,200,255};  
  
/* Assumes that the server reset has been done and a global config entry  
was created successfully in config_cb. And the broadcast address was also  
added to the configuration block */  
  
/* Copy the broadcast address into a structure. */  
memcpy(&broadcast_addr,DHCP_Broadcast_Addr,IP_ADDR_LEN);  
/* Delete the broadcast address from the global control block */  
status = DHCP Delete Broadcast Address(config_cb, NU_NULL,  
                                     &broadcast_addr);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_Broadcast\\_Address](#)

[DHCPS\\_Get\\_Broadcast\\_Address](#)



## DHCP Delete Config Entry

This function removes the configuration structure that is denoted by the variable `config_cb` from the control block link list. The function is responsible for de-allocating the memory for the structure and removing the configuration structure link list.

### Usage

```
STATUS *DHCP Delete Config Entry (DHCP_CB *config_cb);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that is being removed.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; the specified configuration structure has been removed.
- `NU_INVALID_PARM`  
The specified configuration structure was not found in the configuration link list.
- `DHCPERR_PARAMETER_NOT_REMOVED`  
An error occurred while removing the control block from the link list. The control block is still on the link list and still must be deleted.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Delete the config entry */  
status = DHCP Delete Config Entry (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCP Create Config Entry](#)

[DHCP Get Config Entry](#)

## DHCP\_Delete\_DNS\_Server

This function removes a DNS server from the array of DNS server IPv4 addresses.

### Usage

```
STATUS DHCP_Delete_DNS_Server (DHCP_CB          *config_cb,  
                               struct id_struct *client_ip_addr,  
                               struct id_struct *dns_ip_address);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the DNS server will be removed.
- **client\_ip\_addr**  
If this DNS server is meant for only one particular IPv4 address, set this to that address. Otherwise, set this to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- **dns\_ip\_address**  
Pointer to the structure containing the IPv4 address of the DNS server. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; the specified DNS server has been removed from the specified configuration structure.
- **NU\_INVALID\_PARM**  
One of the parameters was invalid. The specified DNS server was not removed.
- **DHCPERR\_PARAMETER\_NOT\_FOUND**  
The DNS server IPv4 address was not located inside the specified configuration control block. The DNS server was not removed..

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
struct id_struct dns_ip_addr;  
CHAR           DHCP_DNS_Server[] = {200,100,200,2};  
  
/* Assumes that the server reset has been done and a global config entry  
was created successfully in config_cb. And the DNS address was also added  
to the configuration block */  
  
/* Copy the DNS address into a structure. */  
memcpy(&dns_ip_addr, DHCP_DNS_Server, IP_ADDR_LEN);  
  
/* Delete the DNS address from the global control block. */
```

```
status = DHCPS_Delete_DNS_Server(config_cb, NU_NULL,  
                                &dns_ip_addr);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_DNS\\_Server](#)

[DHCPS\\_Get\\_DNS\\_Servers](#)

## DHCP Delete\_Domain\_Name

This function removes the domain name from the specified configuration control block.

### Usage

```
STATUS DHCP Delete_Domain_Name (DHCP_CB      *config_cb,  
                                struct id_struct *client_ip_addr,  
                                CHAR            domain_name);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the DNS domain name will be removed.
- **client\_ip\_addr**  
Pointer to the structure containing the IPv4 address of the binding that the domain name is associated with. If the domain name is not binding specific, set this argument to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- **domain\_name**  
Pointer to the buffer that contains the domain name to be deleted.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the DNS domain name has been removed from the specified configuration structure.
- **NU\_INVALID\_PARM**  
Either the config\_cb or domain\_name pointers were set to NU\_NULL.
- **DHCPERR\_PARAMETER\_NOT\_FOUND**  
The specified DNS domain name was not found in specified configuration control block.

### Example

```
STATUS      status;  
CHAR        *domain_name = "mydomain.com";  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a global config entry  
was created successfully in config_cb */  
.....  
  
/* Delete the domain name from the global control block*/  
status = DHCP Delete_Domain_Name(config_cb, NU_NULL, domain_name);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCP Add\\_Domain Name](#)

DHCPS\_Get\_Domain\_Name

## DHCP Delete IP Range

This function removes binding structures for each of the IPv4 addresses in the range between the `ip_low` and `ip_high`. The function de-allocates the memory for the binding structures and removes the binding structure from the link list.

### Usage

```
STATUS DHCP Delete_IP_Range (DHCP_CONFIG_CB *config_cb,  
                             DHCP_CLIENT_ID *client_id,  
                             struct id_struct *ip_low,  
                             struct id_struct *ip_high);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block associated with the IPv4 address range.
- `client_id`  
Pointer to the structure that contains the client identifier of the incoming client in case of static binding only. Set to `NU_NULL` otherwise. Data structure `DHCP_CLIENT_ID` is defined in the [Related Topics](#) section of this chapter.
- `ip_low`  
Pointer to the structure that contains the lowest numerical IPv4 address of the range. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `ip_high`  
Pointer to the structure that contains the highest numerical IPv4 address of the range. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; each of the IPv4 addresses in the range was removed.
- `NU_INVALID_PARM`  
The `config_cb` pointer was invalid. No IP bindings were removed.
- `DHCPERR_IP_RANGE_INVALID`  
The IPv4 range was not contiguous in the binding link list. No IP bindings were removed.
- `DHCPERR_BINDING_NOT_REMOVED`  
An error occurred while removing an IP binding from the link list. A portion of the specified range may have been removed. Use the function `DHCP_Get_IP_Range` to determine which addresses have been removed.
- `DHCP_RANGE_SUBNET_INVALID`  
All of the IPv4 addresses in the specified range cannot be rectified to be on the same subnet with the available subnet masks. None of the IP bindings were removed from the link list.

## Example

```

STATUS          status;
DHCPS_CONFIG_CB *config_cb;
CHAR            IP_Range_Low[] = {200, 100, 200, 30};
CHAR            IP_Range_High[] = {200, 100, 200, 40};
struct id_struct ip_low, ip_high;

/* Assumes that the server reset has been done and a config
entry was created successfully in config_cb. The ip ranges
were also added */
.....

/* Copy the IP range high and low into id_structs. */
memcpy(&ip_low, IP_Range_Low, IP_ADDR_LEN);
memcpy(&ip_high, IP_Range_High, IP_ADDR_LEN);

/* Delete the IP range from the control block. */
status = DHCPS_Delete_IP_Range(config_cb, NU_NULL, &ip_low, &ip_high);

```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_IP\\_Range](#)

[DHCPS\\_Get\\_IP\\_Range](#)

## DHCP Delete\_Router

This function removes a router from the array of router IPv4 addresses.

### Usage

```
STATUS DHCP_Delete_Router (DHCP_CB      *config_cb,  
                           struct id_struct *client_ip_address,  
                           struct id_struct *router_address);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the router will be removed.
- **client\_ip\_address**  
If this router is meant for only one particular IPv4 address, set this to that address. Otherwise, set this to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- **router\_address**  
Pointer to the structure containing the IPv4 address of the router. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter. [id\\_struct](#)

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; the specified router has been removed from the specified configuration structure.
- **NU\_INVALID\_PARM**  
One of the passed in parameters was invalid. The specified router was not removed.
- **DHCP\_PARAMETER\_NOT\_FOUND**  
The router IPv4 address was not located inside the specified configuration control block. The router was not removed.

### Example

```
STATUS      status;  
DHCP_CONFIG_CB *config_cb;  
struct id_struct router_address;  
CHAR        DHCP_Router[] = {200,100,200,1};  
  
/* Assumes that the server reset has been done and a global config entry  
was created successfully in config_cb. And the router address was also  
added to the configuration block */  
  
/* Copy the router address into a structure. */  
memcpy(&router_address, DHCP_Router, IP_ADDR_LEN);  
  
/* Delete the router from the global control block. */
```



```
status = DHCPS_Delete_Router(config_cb, NU_NULL, &router_address);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_Router](#)

[DHCPS\\_Get\\_Router](#)

## DHCPS\_Disable\_Configuration

This function clears the status flag bit (CONFIGURATION\_ENABLED) that tells the Nucleus DHCP Server that the specified configuration control block is now ready to service clients.

### Usage

```
STATUS DHCPS_Disable_Configuration (DHCPS_CB *config_cb);
```

### Arguments

- `config_cb`  
Pointer to the configuration block that is to be disabled.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the DHCP server has been disabled.
- `NU_INVALID_PARM`  
One of the passed in parameters was invalid.
- `DHCPSERR_CONFIG_NOT_FOUND`  
The specified `config_cb` was not located in the list of configuration control block pointers. Ensure that the control block pointer is correct and re-attempt to disable the control block. The enable status flag was not changed.

### Description

Once the configuration is disabled, no leases can be granted for the IP bindings associated with the disabled control block. A configuration control block should be disabled when changes are being made to the control block parameters.

A control block is automatically disabled when it is created.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Disable the config entry */  
status = DHCPS_Disable_Configuration (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Enable\\_Configuration](#)

[DHCPS\\_Create\\_Config\\_Entry](#)

[DHCPS\\_Delete\\_Config\\_Entry](#)

DHCPS\_Get\_Config\_Entry

## DHCPS\_Disable\_Ethernet\_IEEE\_Encapsulation

This function clears the status flag (ETHERNET\_IEEE\_802\_ENCAPSULATION) of the specified configuration control block. This way the DHCP server informs the requesting clients not to enable IEEE 802.3 Ethernet encapsulation.

### Usage

```
STATUS DHCPS_Disable_Ethernet_IEEE_Encapsulation (  
                                                DHCPS_CONFIG_CB *config_cb);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block that the encapsulation status flag must be reset.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the encapsulation status flag has been cleared.
- **NU\_INVALID\_PARM**  
The config\_cb pointer was invalid. The encapsulation status flag has not been altered.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Disable the IEEE 802.3 Ethernet encapsulation */  
status = DHCPS_Disable_Ethernet_IEEE_Encapsulation (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Enable\\_Ethernet\\_IEEE\\_Encapsulation](#)

## DHCPSP\_Disable\_IP\_Forwarding

This function clears the status flag (IP\_FORWARDING\_ENABLED) of the specified configuration control block. This way the DHCP server informs the requesting clients not to enable IPv4 forwarding.

### Usage

```
STATUS DHCPSP_Disable_IP_Forwarding (DHCPSP_CONFIG_CB *config_cb);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the IPv4 time-to-live value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the enable IPv4 forwarding status flag has been cleared.
- `NU_INVALID_PARM`  
The `config_cb` pointer was invalid. The IPv4 forwarding status flag has not been altered.

### Example

```
STATUS          status;  
DHCPSP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Disable the IP forwarding */  
status = DHCPSP_Disable_IP_Forwarding (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPSP\\_Enable\\_IP\\_Forwarding](#)

## DHCPS\_Disable\_TCP\_Keepalive\_Garbage

This function clears the TCP\_KEEPALIVE\_GARBAGE status flag for the specified configuration control block designated by the config\_cb variable.

### Usage

```
DHCPS_Disable_TCP_Keepalive_Garbage (DHCPS_CONFIG_CB *config_cb);
```

### Arguments

- config\_cb  
Pointer to the configuration control block that contains the TCP keepalive garbage flag to be reset.

### Return Values

- NU\_SUCCESS  
Function completed successfully; the TCP keepalive garbage status flag has been cleared.
- NU\_INVALID\_PARM  
The config\_cb pointer was invalid.

### Description

If the TCP keepalive garbage status bit is reset, then requesting clients are informed not to send a garbage byte in keepalive segments.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Clear the TCP keepalive garbage status flag in config_cb*/  
status = DHCPS_Disable_TCP_Keepalive_Garbage (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Enable\\_TCP\\_Keepalive\\_Garbage](#)

## DHCPs\_Disable\_Trailer\_Encapsulation

This function clears the TRAILER\_ENCAPSULATION status flag of the specified configuration control block. This way the DHCP server informs the requesting clients not to use trailer encapsulation.

### Usage

```
STATUS DHCPs_Disable_Trailer_Encapsulation (DHCPs_CONFIG_CB *config_cb);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block that contains the enable trailer encapsulation status flag to be reset.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the enable trailer encapsulation status flag has been cleared.
- **NU\_INVALID\_PARM**  
The config\_cb pointer was invalid. The enable trailer encapsulation status flag has not been altered.

### Example

```
STATUS          status;  
DHCPs_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Disable Trailer Encapsulation */  
status = DHCPs_Disable_Trailer_Encapsulation (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPs\\_Enable\\_Trailer\\_Encapsulation](#)

## DHCPS\_Enable\_Configuration

This function sets the CONFIGURATION\_ENABLED status flag bit that tells the Nucleus DHCP Server that the specified configuration control block is now ready to service clients.

### Usage

```
STATUS DHCPS_Enable_Configuration (DHCPS_CB *config_cb);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that is to be enabled.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the DHCP server has been enabled.
- `NU_INVALID_PARM`  
The `config_cb` pointer was invalid.
- `DHCPSERR_CONFIG_NOT_FOUND`  
The specified `config_cb` was not located in the list of configuration control block pointers. Ensure that the control block pointer is correct and re-attempt to enable the control block. The enable status flag was unchanged.

### Description

If the configuration control block is not enabled, no leases can be granted for the IPv4 bindings associated with the disabled control block.

A control block is automatically disabled when it is created.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Enable the config entry */  
status = DHCPS_Enable_Configuration (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Config\\_Entry](#)

[DHCPS\\_Create\\_Config\\_Entry](#)

[DHCPS\\_Delete\\_Config\\_Entry](#)



## DHCPS\_Enable\_Ethernet\_IEEE\_Encapsulation

This function sets the `ETHERNET_IEEE_802_ENCAPSULATION` status flag for the specified configuration control block that informs clients to enable IEEE 802.3 Ethernet encapsulation.

### Usage

```
STATUS DHCPS_Enable_Ethernet_IEEE_Encapsulation (
    DHCPS_CONFIG_CB *config_cb);
```

### Arguments

- `config_cb`  
 Pointer to the configuration control block where the Ethernet encapsulation status flag will be set.

### Return Values

- `NU_SUCCESS`  
 Function completed successfully; the Ethernet encapsulation enabled status flag has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
 The `config_cb` pointer was not valid. The encapsulation enable flag was not set.

### Description

This option cannot be implemented for a specific binding. Therefore, Ethernet encapsulation is enabled for each of the IPv4 bindings associated with the specified configuration control block.

### Example

```
STATUS          status;
DHCPS_CONFIG_CB *config_cb;

/* Assumes that the server reset has been done and a config entry was
   created successfully in config_cb */
....

/* Enable the IEEE 802.3 Ethernet encapsulation */
status = DHCPS_Enable_Ethernet_IEEE_Encapsulation (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Disable\\_Ethernet\\_IEEE\\_Encapsulation](#)

## DHCPS\_Enable\_IP\_Forwarding

This function sets the `IP_FORWARDING_ENABLED` status flag for the specified configuration control block that informs clients to enable IPv4 forwarding.

### Usage

```
STATUS DHCPS_Enable_IP_Forwarding (DHCPS_CONFIG_CB *config_cb);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the IP forwarding status flag will be set.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; the IPv4 forwarding enabled status flag has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
The `config_cb` pointer was not valid. The IPv4 forwarding status flag was not set.

### Description

This option cannot be implemented for a specific binding. Therefore, IPv4 forwarding is enabled for each of the IPv4 bindings associated with the specified configuration control block.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Enable the IP forwarding */  
status = DHCPS_Enable_IP_Forwarding (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Disable\\_IP\\_Forwarding](#)

## DHCPSP\_Enable\_TCP\_Keepalive\_Garbage

This function sets the TCP\_KEEPALIVE\_GARBAGE status flag for the specified configuration control block designated by the config\_cb variable.

### Usage

```
STATUS DHCPSP_Enable_TCP_Keepalive_Garbage (DHCPSP_CONFIG_CB *config_cb);
```

### Arguments

- config\_cb  
Pointer to the configuration control block where the TCP keepalive garbage status flag will be set.

### Return Values

- NU\_SUCCESS  
Function completed successfully; the TCP keepalive garbage flag has been set in the specified configuration structure.
- NU\_INVALID\_PARM  
The config\_cb pointer was not valid. The keepalive garbage flag was not set.

### Description

If the TCP keepalive garbage status bit is set, then requesting clients are informed that a garbage byte should be sent in keepalive segments.

### Example

```
STATUS          status;  
DHCPSP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set the TCP keepalive garbage status flag in config_cb*/  
status = DHCPSP_Enable_TCP_Keepalive_Garbage (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPSP\\_Disable\\_TCP\\_Keepalive\\_Garbage](#)

## DHCPS\_Enable\_Trailer\_Encapsulation

This function sets the TRAILER\_ENCAPSULATION status flag for the specified configuration control block that will inform clients to attempt to use trailer encapsulation.

### Usage

```
STATUS DHCPS_Enable_Trailer_Encapsulation (DHCPS_CONFIG_CB *config_cb);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the trailer encapsulation status flag will be set.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; the trailer encapsulation enabled status flag has been added to the specified configuration.
- **NU\_INVALID\_PARM**  
The config\_cb pointer was not valid. The trailer encapsulation enable flag was not set.

### Description

This option cannot be implemented for a specific binding. Therefore, trailer encapsulation will be enabled for each of the IPv4 bindings associated with the specified configuration control block.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Enable Trailer Encapsulation */  
status = DHCPS_Enable_Trailer_Encapsulation (config_cb);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Disable\\_Trailer\\_Encapsulation](#)

## DHCPSP\_Get\_ARP\_Cache\_Timeout

This function copies the IPv4 time-to-live value from the specified configuration control block into the buffer pointed to by `ip_ttl_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPSP_Get_ARP_Cache_Timeout (DHCPSP_CONFIG_CB    *config_cb,  
                                struct id_struct      *client_ip_addr,  
                                UINT8                 *arp_cache_buffer,  
                                INT                   buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the ARP cache timeout value.
- `client_ip_addr`  
If the desired timeout value is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `arp_cache_buffer`  
Pointer to the buffer where the timeout value will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes that were written in the buffer  
Function completed successfully.
- `NU_INVALID_PARM`  
The `config_cb` or the `arp_cache_buffer` or pointer was invalid.
- `DHCPSPERR_BUFFER_TOO_SMALL`  
The `arp_cache_buffer` was too small to hold the timeout value.

### Example

```
INT num_bytes;  
DHCPSP_CONFIG_CB *config_cb;  
UINT8 arp_cache_buffer[4];  
INT buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Get ARP cache timeout value */  
num_bytes = DHCPSP_Get_ARP_Cache_Timeout (config_cb, NU_NULL,
```

```
arp_cache_buffer,  
buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_ARP\\_Cache\\_Timeout](#)

## DHCPSP\_Get\_Broadcast\_Address

This function copies the broadcast address from the specified configuration control block into the buffer pointed to by `broadcast_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPSP_Get_Broadcast_Address (DHCPSP_CONFIG_CB *config_cb,  
                                struct id_struct *client_ip_addr,  
                                UINT8 *broadcast_buffer,  
                                INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block that is associated with this particular configuration structure.
- `client_ip_addr`  
If the desired broadcast address is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `broadcast_buffer`  
Pointer to the buffer where each of the broadcast addresses will be copied.
- `buffer_size`  
The size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
The function completed successfully.
- `NU_INVALID_PARM`  
The `config_cb` or `broadcast_buffer` pointer is invalid.
- `DHCPSPERR_BUFFER_TOO_SMALL`  
The `broadcast_buffer` was too small to hold the broadcast address.

### Example

```
INT num_bytes;  
DHCPSP_CONFIG_CB *config_cb;  
UINT8 broadcast_buffer[4];  
INT buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....
```

```
/* Get the broadcast address */  
num_bytes = DHCPS_Get_Broadcast_Address (config_cb, NU_NULL,  
                                         broadcast_buffer,  
                                         buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_Broadcast\\_Address](#)

[DHCPS\\_Delete\\_Broadcast\\_Address](#)



## DHCPS\_Get\_Config\_Entry

This function copies the control block pointers of each of the configuration structures of the DHCP server into the provided buffer pointed to by config\_buffer.

### Usage

```
INT DHCPS_Get_Config_Entry (UINT8 *config_buffer,  
                           INT     buffer_size);
```

### Arguments

- config\_buffer  
Pointer to the buffer where the configuration control block pointers will be copied.
- buffer\_size  
Size of the buffer in bytes. The buffer should be at least 4 bytes times the number of control blocks that are present.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- NU\_INVALID\_PARM  
The config\_buffer pointer was invalid.
- DHCPSEERR\_BUFFER\_TOO\_SMALL  
The config\_buffer was too small to hold all of the control pointers.

### Example

```
INT     num_bytes;  
UINT8   config_buffer[8]; /* There are two configuration control blocks */  
INT     buffer_size = 8;  
  
/* Get the config entries */  
num_bytes = DHCPS_Get_Config_Entry (config_buffer, buffer_size);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Create\\_Config\\_Entry](#)

[DHCPS\\_Delete\\_Config\\_Entry](#)

## DHCPS\_Get\_Default\_Lease\_Time

This function copies the lease time from the specified configuration control block into the buffer pointed to by `default_lease_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPS_Get_Default_Lease_Time (DHCPS_CONFIG_CB *config_cb,  
                                struct id_struct *client_ip_addr,  
                                UINT8 *default_lease_buffer,  
                                INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block that is associated with this configuration structure.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the lease time will be copied to. If the lease time is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `default_lease_buffer`  
Pointer to the buffer where the lease time will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- Total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `default_lease_buffer` pointer was invalid.
- `DHCPSERR_BUFFER_TOO_SMALL`  
The `default_lease_buffer` was too small to hold the time-to-live value.

### Example

```
INT num_bytes;  
DHCPS_CONFIG_CB *config_cb;  
UINT8 default_lease_buffer[4];  
INT buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....
```

```
/* Get the Default Lease time */  
num_bytes = DHCPS_Get_Default_Lease_Time (config_cb, NU_NULL,  
                                           default_lease_buffer,  
                                           buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Default\\_Lease\\_Time](#)

## DHCPGetDNSServers

This function copies each of the DNS server IPv4 addresses from the specified configuration control block into the buffer pointed to by `dns_server_buffer` and returns the total number of bytes that were written into the buffer.

### Usage

```
INT DHCPGetDNSServers(DHCP_CONFIG_CB *config_cb,  
                      struct id_struct *client_ip_addr,  
                      INT8 *dns_buffer,  
                      INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block that is associated with this configuration structure.
- `client_ip_addr`  
If the domain name servers associated with a specified binding are required, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `dns_server_buffer`  
Pointer to the buffer where the IPv4 addresses of the DNS servers will be placed.
- `Buffer_size`  
Size of buffer in bytes, it should be at least 4 bytes times `DHCP_MAX_DNS_SERVERS`.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `dns_server_buffer` pointer was invalid.
- `DHCPERR_BUFFER_TOO_SMALL`  
The `dns_server_buffer` was too small to hold all the DNS server IPv4 addresses.

### Example

```
INT num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8 dns_buffer[4 * DHCP_MAX_DNS_SERVERS];  
INT buffer_size = 4 * DHCP_MAX_DNS_SERVERS;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb. The DNS servers were also added */  
....
```

```
/* Get the DNS servers list */  
num_bytes = DHCPS_Get_DNS_Servers(config_cb, NU_NULL,  
                                   dns_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_DNS\\_Server](#)

[DHCPS\\_Delete\\_DNS\\_Server](#)

## DHCPs\_Get\_Domain\_Name

This function copies the domain name from the specified configuration control block into the buffer pointed to by domain\_buffer and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPs_Get_Domain_Name (DHCPs_CB      *config_cb,  
                           struct id_struct *client_ip_addr,  
                           UINT8          *domain_buffer,  
                           INT            buffer_size);
```

### Arguments

- config\_cb  
Pointer to the control block that is associated with this configuration structure.
- client\_ip\_addr  
If the desired domain name is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- domain\_buffer  
Pointer to the buffer where each of the DNS domain names will be copied.
- buffer\_size  
Size of the buffer in bytes. The buffer should be at least DHCPs\_MAX\_DOMAIN\_NAME\_LENGTH bytes in size.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- NU\_INVALID\_PARM  
The domain\_buffer pointer was invalid
- DHCPsERR\_BUFFER\_TOO\_SMALL  
The domain\_buffer was too small to hold all the domain name.

### Example

```
INT          num_bytes;  
DHCPs_CONFIG_CB *config_cb;  
UINT8        domain_buffer[DHCPs_MAX_DOMAIN_NAME_LENGTH];  
INT          buffer_size = DHCPs_MAX_DOMAIN_NAME_LENGTH;  
  
/* Assumes that the server reset has been done and a config entry was  
   created successfully in config_cb. The Domain name  
   was also added */  
....
```

```
/* Get the Domain Name */  
num_bytes = DHCPS_Get_Domain_Name (config_cb, NU_NULL,  
                                   domain_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_Domain Name](#)

[DHCPS\\_Delete\\_Domain\\_Name](#)

## DHCP Get\_IP\_Range

This function copies each of the IPv4 addresses from the binding link list of the specified configuration control block into the provided buffer pointed to by `binding_buffer`.

### Usage

```
INT DHCP Get_IP_Range (DHCP_CONFIG_CB *config_cb,  
                      UINT8          *binding_buffer,  
                      INT            buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block from which the IP bindings are copied.
- `binding_buffer`  
Pointer to the buffer where the IPv4 addresses are copied.
- `buffer_size`  
Size of the buffer in bytes. The buffer should be at least 4-bytes times the number of IPv4 addresses added to the server.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `binding_buffer` pointer was invalid
- `DHCPERR_BUFFER_TOO_SMALL`  
The `binding_buffer` was too small to hold all the IPv4 addresses.

### Example

```
INT          num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8        binding_buffer[40]; /* IP range consists  
                                of 10 addresses */  
  
INT          buffer_size = 40;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb. The Domain name  
was also added */  
....  
  
/* Get the IP range */  
num_bytes = DHCP Get_IP_Range (config_cb, binding_buffer,  
                              buffer_size);
```



## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_IP\\_Range](#)

[DHCPS\\_Delete\\_IP\\_Range](#)

## DHCPGet\_IP\_TTL

This function copies the IPv4 time-to-live value from the specified configuration control block into the buffer pointed to by `ip_ttl_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPGet_IP_TTL (DHCP_CONFIG_CB *config_cb,  
                   struct id_struct *client_ip_addr,  
                   INT8 *ip_ttl_buffer,  
                   INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the IPv4 time-to-live value.
- `client_ip_addr`  
If the desired time-to-live value is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `ip_ttl_buffer`  
Pointer to the buffer where the time-to-live value will be copied.
- `buffer_size`  
Size of the buffer in bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARAM`  
The `config_cb` or the `ip_ttl_buffer` pointer was invalid
- `DHCPERR_BUFFER_TOO_SMALL`  
The `binding_buffer` was too small to hold all the IPv4 addresses.

### Example

```
INT num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8 ip_ttl_buffer;  
INT buffer_size = 1;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Get the IP TTL value */  
num_bytes = DHCPGet_IP_TTL (config_cb, NU_NULL,
```

```
&ip_ttl_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_IP\\_TTL](#)

## DHCPSP\_Get\_Offered\_Wait\_Time

This function copies the offered wait time value from the specified configuration control block into the buffer pointed to by `offered_wait_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPSP_Get_Offered_Wait_Time (DHCPSP_CONFIG_CB *config_cb,  
                                struct id_struct *client_ip_addr,  
                                UINT8 *offered_wait_buffer,  
                                INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block associated with this configuration structure.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the offered wait time value will be copied to. If the wait time value is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `offered_wait_buffer`  
Pointer to the buffer where the offered wait time will be copied.
- `buffer_size`  
Size of the buffer in bytes, it must be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARAM`  
The `config_cb` or `offered_wait_buffer` pointer was invalid.
- `DHCPSP_BUFFER_TOO_SMALL`  
The `offered_wait_buffer` was too small to hold the time-to-live value.

### Example

```
INT num_bytes;  
DHCPSP_CONFIG_CB *config_cb;  
UINT8 offered_wait_buffer[4];  
INT buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....
```

```
/* Get the Offered wait time value */  
num_bytes = DHCPS_Get_Offered_Wait_Time (config_cb, NU_NULL,  
                                         offered_wait_buffer,  
                                         buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Offered\\_Wait\\_Time](#)

## DHCPSP\_Get\_Rebind\_Time

This function copies the lease rebinding time from the specified configuration control block into the buffer pointed to by `rebind_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPSP_Get_Rebind_Time (DHCPSP_CONFIG_CB *config_cb,  
                           struct id_struct *client_ip_addr,  
                           UINT8 *rebind_buffer,  
                           INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block associated with this configuration structure.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the lease rebinding time will be copied to. If the rebinding time is not binding-specific, set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `rebind_buffer`  
Pointer to the buffer where the lease rebinding time will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARAM`  
The `config_cb` or `rebind_buffer` pointer was invalid.
- `DHCPSPERR_BUFFER_TOO_SMALL`  
The `rebind_buffer` was too small to hold the time-to-live value.

### Example

```
INT num_bytes;  
DHCPSP_CONFIG_CB *config_cb;  
UINT8 rebind_buffer[4];  
INT buffer_size = 4;  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....
```

```
/* Get the Rebind time value */  
num_bytes = DHCPS_Get_Rebind_Time (config_cb, NU_NULL,  
                                   rebind_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Rebind\\_Time](#)

## DHCPS\_Get\_Renewal\_Time

This function copies the lease renewal time from the specified configuration control block into the buffer pointed to by `renew_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPS_Get_Renewal_Time (DHCPS_CONFIG_CB *config_cb,  
                           struct id_struct *client_ip_addr,  
                           UINT8 *renew_buffer,  
                           INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block associated with this configuration structure.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the lease renewal time will be copied to. If the renewal time is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `renew_buffer`  
Pointer to the buffer where the lease renewal time will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `renew_buffer` pointer was invalid.
- `DHCPSERR_BUFFER_TOO_SMALL`  
The `renew_buffer` was too small to hold the time-to-live value.

### Example

```
INT num_bytes;  
DHCPS_CONFIG_CB *config_cb;  
UINT8 renew_buffer[4];  
INT buffer_size = 4;  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....
```



```
/* Get the Renewal time value */  
num_bytes = DHCPS_Get_Renewal_Time (config_cb, NU_NULL,  
                                     renew_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Renewal\\_Time](#)

## DHCPS\_Get\_Router

This function copies each of the router IPv4 addresses from the specified configuration control block into the buffer pointed to by `router_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPS_Get_Router (DHCPS_CB      *config_cb,  
                      struct id_struct *client_ip_addr,  
                      UINT8          *router_buffer,  
                      INT            buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the control block associated with this configuration structure.
- `client_ip_addr`  
If the router addresses associated with a specified binding are required, set this argument to the IPv4 address of that binding. Otherwise, set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `router_buffer`  
Pointer to the buffer where the IPv4 addresses of the routes will be copied.
- `buffer_size`  
Size of the buffer in bytes. The buffer should be at least 4 bytes times `DHCPS_MAX_ROUTERS`.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `router_buffer` pointer was invalid.
- `DHCPSERR_BUFFER_TOO_SMALL`  
The `router_buffer` was too small to hold the time-to-live value.

### Example

```
INT          num_bytes;  
DHCPS_CONFIG_CB *config_cb;  
UINT8        router_buffer[4 * DHCPS_MAX_ROUTERS];  
INT          buffer_size = 4 * DHCPS_MAX_ROUTERS;  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb. The routers were also added */  
....
```

```
/* Get the routers' list of IP addresses */  
num_bytes = DHCPS_Get_Router (config_cb, NU_NULL,  
                             router_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Add\\_Router](#)

[DHCPS\\_Delete\\_Router](#)

## DHCPS\_Get\_Subnet\_Address

This function copies the subnet address from the specified configuration control block into the buffer pointed to by `subnet_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPS_Get_Subnet_Address (const DHCP_CONFIG_CB *config_cb,  
                             struct id_struct      *client_ip_addr,  
                             UINT8                 *subnet_buffer,  
                             INT                   buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the subnet address.
- `client_ip_addr`  
If the desired subnet address is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `subnet_buffer`  
Pointer to the buffer where the subnet address will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `subnet_buffer` pointer was invalid.
- `DHCPSERR_BUFFER_TOO_SMALL`  
The `subnet_buffer` was too small to hold the subnet address.

### Example

```
INT          num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8        subnet_buffer[4];  
INT          buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
   created successfully in config_cb. The subnet address was also added */  
....  
  
/* Get the Subnet address */  
num_bytes = DHCPS_Get_Subnet_Address (config_cb, NU_NULL,  
                                     subnet_buffer,
```

```
buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Subnet\\_Address](#)

[DHCPS\\_Set\\_Subnet\\_Mask](#)

[DHCPS\\_Get\\_Subnet\\_Mask](#)

## DHCPGet\_Subnet\_Mask

This function copies the subnet mask from the specified configuration control block into the buffer pointed to by `subnet_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPGet_Subnet_Mask (const DHCP_CONFIG_CB *config_cb,  
                        struct id_struct      *client_ip_addr,  
                        UINT8                 *subnet_buffer,  
                        INT                   buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the subnet mask.
- `client_ip_addr`  
If the desired subnet mask is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `subnet_buffer`  
Pointer to the buffer where the subnet mask will be copied.
- `buffer_size`  
Size of the buffer in bytes, it should be at least 4 bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `subnet_buffer` pointer was invalid.
- `DHCPERR_BUFFER_TOO_SMALL`  
The `subnet_buffer` was too small to hold the subnet mask value.

### Example

```
INT          num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8        subnet_buffer[4];  
INT          buffer_size = 4;  
  
/* Assumes that the server reset has been done and a config entry was  
   created successfully in config_cb. The subnet mask was also added */  
....  
  
/* Get the Subnet mask */  
num_bytes = DHCPGet_Subnet_Mask (config_cb, NU_NULL,  
                                subnet_buffer,
```

```
buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_Subnet\\_Address](#)

[DHCPS\\_Get\\_Subnet\\_Address](#)

[DHCPS\\_Set\\_Subnet\\_Mask](#)

## DHCPGet\_TCP\_KeepaliveInterval

This function copies the TCP keepalive interval value from the specified configuration control block into the buffer pointed to by `tcp_keepalive_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPGet_TCP_KeepaliveInterval (
                                DHCP_CONFIG_CB *config_cb,
                                struct id_struct *client_ip_addr,
                                UINT8          *tcp_keepalive_buffer,
                                INT            buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the TCP keepalive value.
- `client_ip_addr`  
If the desired keepalive value is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `tcp_keepalive_buffer`  
Pointer to the buffer where the keepalive value will be copied.
- `buffer_size`  
Size of the buffer in bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `tcp_keepalive_buffer` pointer was invalid.
- `DHCPERR_BUFFER_TOO_SMALL`  
The `tcp_keepalive_buffer` was too small to hold the subnet mask value.

### Example

```
INT          num_bytes;
DHCP_CONFIG_CB *config_cb;
UINT8       tcp_keepalive_buffer;
INT         buffer_size = 1;

/* Assumes that the server reset has been done and a config entry was
created successfully in config_cb */
....
```



```
/* Get the TCP Keepalive Interval value */  
num_bytes = DHCPS_Get_TCP_Keepalive_Interval (config_cb, NU_NULL,  
                                              &tcp_keepalive_buffer,  
                                              buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_TCP\\_Keepalive\\_Interval](#)

## DHCPGet\_TCP\_TTL

This function copies the TCP time-to-live value from the specified configuration control block into the buffer pointed to by `tcp_ttl_buffer` and returns the total number of bytes written into the buffer.

### Usage

```
INT DHCPGet_TCP_TTL (DHCP_CONFIG_CB *config_cb,  
                    struct id_struct *client_ip_addr,  
                    UINT8 *tcp_ttl_buffer,  
                    INT buffer_size);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block that contains the TCP time-to-live value.
- `client_ip_addr`  
If the desired time-to-live value is associated with a specified binding, set this argument to the IPv4 address of that binding. Otherwise, set it to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter. `id_struct`
- `tcp_ttl_buffer`  
Pointer to the buffer where the time-to-live value will be copied.
- `buffer_size`  
Size of the buffer in bytes.

### Return Values

- The total number of bytes written into the buffer  
On successful completion.
- `NU_INVALID_PARM`  
The `config_cb` or `tcp_ttl_buffer` pointer was invalid.
- `DHCPERR_BUFFER_TOO_SMALL`  
The `tcp_ttl_buffer` was too small to hold the subnet mask value.

### Example

```
INT num_bytes;  
DHCP_CONFIG_CB *config_cb;  
UINT8 tcp_ttl_buffer;  
INT buffer_size = 1;  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Get the TCP TTL value */  
num_bytes = DHCPGet_TCP_TTL (config_cb, NU_NULL,  
                             &tcp_ttl_buffer, buffer_size);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Set\\_TCP\\_TTL](#)

## DHCPSet\_ARP\_Cache\_Timeout

This function sets the ARP cache timeout value for the specified configuration control block designated by the `config_cb` variable.

### Usage

```
STATUS DHCPSet_ARP_Cache_Timeout (DHCP_CONFIG_CB *config_cb,  
                                  struct id_struct *client_ip_addr,  
                                  UINT32          arp_cache_timeout);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the ARP cache timeout will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the ARP cache timeout value will be added to. If the time-to-live value is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `arp_cache_timeout`  
ARP cache timeout value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the cache timeout value has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
The `config_cb` was invalid. The timeout value was not added.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set ARP cache timeout value to 300 seconds */  
status = DHCPSet_ARP_Cache_Timeout (config_cb, NU_NULL, 300);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPGet\\_ARP\\_Cache\\_Timeout](#)

## DHCPSetDefaultLeaseTime

This function sets the default lease time for the specified configuration control block designated by the `config_cb` variable.

### Usage

```
STATUS DHCPSetDefaultLeaseTime(DHCP_CONFIG_CB *config_cb,  
                               struct id_struct *client_ip_addr,  
                               UINT32          default_lease_time);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the lease time will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the lease time will be added to. If the lease time is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `default_lease_time`  
Lease time value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the lease time has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
The `config_cb` was invalid. The lease time was not added

### Description

This value must be set either in a specific configuration control block or the global control block. The server does not have a default value that is used.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set Default Lease Time value to 300 seconds */  
status = DHCPSetDefaultLeaseTime (config_cb, NU_NULL, 300);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Default\\_Lease\\_Time](#)

## DHCPSet\_IP\_TTL

This function sets the IPv4 time-to-live value for the specified configuration control block designated by the `config_cb` variable.

### Usage

```
STATUS DHCPSet_IP_TTL (DHCP_CONFIG_CB *config_cb,  
                      struct id_struct *client_ip_addr,  
                      UINT8 ip_ttl);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the IPv4 time-to-live value will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the IPv4 time-to-live will be added to. If the time-to-live is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `ip_ttl`  
IPv4 time-to-live value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the time-to-live has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
One of the passed in parameters was invalid. The time-to-live was not added.

### Example

```
STATUS status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set IP TTL value to 100 */  
status = DHCPSet_IP_TTL (config_cb, NU_NULL, 100);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPGet\\_IP\\_TTL](#)

## DHCPSet\_Offered\_Wait\_Time

This function sets the amount of time that an offered binding is protected from being re-offered to a different requesting client.

### Usage

```
STATUS DHCPSet_Offered_Wait_Time (DHCP_CONFIG_CB *config_cb,  
                                struct id_struct *client_ip_addr,  
                                UINT32          offered_wait_time);
```

### Arguments

- **config\_cb**  
Pointer to the configuration control block where the offered wait time will be placed.
- **client\_ip\_addr**  
Pointer to the structure that contains the IPv4 address of the specific binding that the offered wait time will be added to. If the wait time is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- **offered\_wait\_time**  
Offered wait time value.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; the offered wait time has been added to the specified configuration structure.
- **NU\_INVALID\_PARM**  
The `config_cb` pointer was invalid. The offered wait time was not added.

### Description

The offered wait time is stored in the binding structure and is decremented by a lease timer until the binding is requested by the original client or the offered wait time expires. If the offered wait time is for a specific binding, then the argument `client_ip_addr` should be set to the IPv4 address of the binding. Otherwise, `client_ip_addr` should be set to `NU_NULL`.

This value must be set either in a specific configuration control block or the global control block. The server does not have a default value.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set Offered wait time to 150 seconds */
```



```
status = DHCPS_Set_Offered_Wait_Time (config_cb, NU_NULL, 150);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Offered\\_Wait\\_Time](#)

## DHCPSet\_Rebind\_Time

This function sets the lease rebinding time (T2 time) for the specified configuration control block, designated by the `config_cb` variable.

### Usage

```
STATUS DHCPSet_Rebind_Time (DHCP_CONFIG_CB *config_cb,  
                           struct id_struct *client_ip_addr,  
                           UINT32          rebind_time);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the rebinding time will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding. The lease rebinding time will be set in this structure. If the rebinding time is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `rebind_time`  
Lease rebinding time value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the lease rebinding time has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
The `config_cb` pointer was invalid. The rebinding time was not added.

### Description

This value must be set either in a specific configuration control block or the global control block. The server does not have a default value.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set Rebind time to 225 seconds */  
status = DHCPSet_Rebind_Time (config_cb, NU_NULL, 225);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Rebind\\_Time](#)

## DHCPSet\_Renewal\_Time

This function sets the lease renewal time (T1 time) for the specified configuration control block designated by the config\_cb variable.

### Usage

```
STATUS DHCPSet_Renewal_Time (DHCP_CONFIG_CB *config_cb,  
                             struct id_struct *client_ip_addr,  
                             UINT32          renew_time);
```

### Arguments

- config\_cb  
Pointer to the configuration control block where the renewal time will be placed.
- client\_ip\_addr  
Pointer to the structure that contains the IPv4 address of the specific binding that the lease renewal time will be added to. If the renewal time is not binding-specific, then set this argument to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- renew\_time  
Lease renewal time value.

### Return Values

- NU\_SUCCESS  
The function completed successfully; the lease renewal time has been added to the specified configuration structure.
- NU\_INVALID\_PARM  
The config\_cb pointer was invalid. The renewal time was not added.

### Description

This value must be set either in a specific configuration control block or the global control block. The server does not have a default value.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set Renewal time to 150 seconds */  
status = DHCPSet_Renewal_Time (config_cb, NU_NULL, 150);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Renewal\\_Time](#)

## DHCPSet\_Subnet\_Address

This function sets the subnet address in the specified configuration control block designated by the config\_cb variable.

### Usage

```
STATUS DHCPSet_Subnet_Address (DHCP_CB      *config_cb,  
                              struct id_struct *client_ip_addr,  
                              struct id_struct *subnet_addr);
```

### Arguments

- config\_cb  
Pointer to the configuration control block where the subnet address will be placed.
- client\_ip\_addr  
Pointer to the structure that contains the IPv4 address of the specific binding that the subnet address will be added to. If the subnet address is not binding-specific, then set this argument to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- subnet\_addr  
Pointer to the structure that contains the subnet address. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The function completed successfully; the subnet address has been added to the specified configuration structure.
- NU\_INVALID\_PARM  
Either the config\_cb or the subnet\_addr pointers were set to NU\_NULL.

### Description

A subnet address must be declared for either the specific configuration control block or must be placed in the global configuration control block. The Nucleus DHCP Server will not function if a subnet address has not been declared.

### Example

```
STATUS      status;  
struct id_struct subnet_addr;  
DHCP_CONFIG_CB *config_cb;  
CHAR      Subnet_Address[] = {200,100,200,0};  
  
/* Assumes that the server reset has been done and a config entry was  
   created successfully in config_cb */  
.....  
  
/* Copy the subnet address into a structure. */
```

```
memcpy(&subnet_addr, Subnet_Address, IP_ADDR_LEN);

/* Add the subnet address to the control block. */
status = DHCPS_Set_Subnet_Address (config_cb, NU_NULL,
                                   subnet_addr);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Subnet\\_Address](#)

[DHCPS\\_Get\\_Subnet\\_Mask](#)

[DHCPS\\_Set\\_Subnet\\_Mask](#)

## DHCPSet\_Subnet\_Mask

This function sets the subnet mask in the specified configuration control block designated by the config\_cb variable.

### Usage

```
STATUS DHCPSet_Subnet_Mask (DHCP_CONFIG_CB *config_cb,  
                           struct id_struct *client_ip_addr,  
                           struct id_struct *subnet_mask);
```

### Arguments

- config\_cb  
Pointer to the configuration control block where the subnet mask will be placed.
- client\_ip\_addr  
Pointer to the structure that contains the IPv4 address of the specific binding that the subnet mask will be added to. If the subnet mask is not binding-specific, then set this argument to NU\_NULL. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.
- subnet\_mask  
Pointer to the structure that contains the subnet mask. Data structure [id\\_struct](#) is defined in the [Related Topics](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The function completed successfully; the subnet mask has been added to the specified configuration structure.
- NU\_INVALID\_PARM  
Either the config\_cb or the subnet\_mask pointers were set to NU\_NULL.

### Description

A subnet mask must be declared for either the specific configuration control block or the global configuration control block. The Nucleus DHCP Server does not function if a subnet mask has not been declared.

### Example

```
STATUS          status;  
struct id_struct subnet_mask;  
DHCP_CONFIG_CB *config_cb;  
CHAR            Subnet_Mask[] = {200,100,200,0};  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
.....  
  
/* Copy the subnet mask into a structure. */  
memcpy(&subnet_mask, Subnet_Mask, IP_ADDR_LEN);
```



```
/* Add the subnet mask to the control block. */  
status = DHCPS_Set_Subnet_Address (config_cb, NU_NULL, &subnet_mask);
```

## Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_Subnet\\_Mask](#)

[DHCPS\\_Get\\_Subnet\\_Address](#)

[DHCPS\\_Set\\_Subnet\\_Address](#)

## DHCPS\_Set\_TCP\_Keepalive\_Interval

This function sets the TCP keepalive interval (in seconds) for the specified configuration control block designated by the `config_cb` variable.

### Usage

```
STATUS DHCPS_Set_TCP_Keepalive_Interval(DHCPS_CONFIG_CB *config_cb,  
                                         struct id_struct *client_ip_addr,  
                                         UINT8 tcp_keepalive);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the IPv4 keepalive interval value will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the TCP keepalive interval will be added to. If the keepalive interval is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `tcp_keepalive`  
TCP keepalive interval (in seconds). If this value is set to zero, requesting clients will be informed not to use TCP keepalive.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the keepalive interval has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
One of the passed in parameters was invalid. The keepalive interval was not added.

### Example

```
STATUS          status;  
DHCPS_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set TCP Keepalive Interval to 100 */  
status = DHCPS_Set_TCP_Keepalive_Interval (config_cb, NU_NULL, 100);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Get\\_TCP\\_Keepalive\\_Interval](#)

## DHCPSet\_TCP\_TTL

This function sets the TCP time-to-live value for the specified configuration control block designated by the `config_cb` variable.

### Usage

```
STATUS DHCPSet_TCP_TTL (DHCP_CONFIG_CB *config_cb,  
                        struct id_struct *client_ip_addr,  
                        UINT8           tcp_ttl);
```

### Arguments

- `config_cb`  
Pointer to the configuration control block where the IPv4 time-to-live value will be placed.
- `client_ip_addr`  
Pointer to the structure that contains the IPv4 address of the specific binding that the TCP time-to-live value will be added to. If the time-to-live value is not binding-specific, then set this argument to `NU_NULL`. Data structure `id_struct` is defined in the [Related Topics](#) section of this chapter.
- `tcp_ttl`  
TCP time-to-live value.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the time-to-live has been added to the specified configuration structure.
- `NU_INVALID_PARM`  
One of the passed in parameters was invalid. The time-to-live value was not added.

### Example

```
STATUS          status;  
DHCP_CONFIG_CB *config_cb;  
  
/* Assumes that the server reset has been done and a config entry was  
created successfully in config_cb */  
....  
  
/* Set TCP TTL value to 100 */  
status = DHCPSet_TCP_TTL (config_cb, NU_NULL, 100);
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPGet\\_TCP\\_TTL](#)

## DHCPS\_Shutdown\_Server

This function shuts down the Nucleus DHCP Server. The function saves the configuration and binding data to their respective disk files and de-allocates all memory the server may have allocated.

### Usage

```
STATUS DHCPS_Shutdown_Server();
```

### Arguments

- None

### Return Values

- **NU\_SUCCESS**  
Function completed successfully; the DHCP server has been shutdown.
- **DHCPERR\_SERVER\_NOT\_SHUTDOWN**  
An error occurred during the shutting down of the DHCP server.

### Example

```
STATUS status;  
  
/* Shutdown the DHCP Server */  
status = DHCPS_Shutdown_Server();
```

### Related Topics

[DHCP Server API Function Reference](#)

[DHCPS\\_Server\\_Reset](#)

































































# Chapter 6

## Security Sockets Layer (SSL)

---

SSL is used for secure communications over the Internet and is based on a certificate process that verifies either the client or server is who they say they are. Once this verification is complete, all information passed between the client and server is encrypted. SSL is the most widely used form of encryption on the Internet today. It is automatically built into many Web browsers.

OpenSSL is very large and requires a great amount of memory for its networking, encryption and session cache support. Therefore Nucleus SSL is based on OpenSSL, but is optimized for use in embedded applications.

CyaSSL is a an alternative library to OpenSSL. CyaSSL is currently used with [HTTP Lite](#) and [WebSockets](#) to provide secure connectivity. Since OpenSSL and CyaSSL share a similar API, they cannot be enabled simultaneously. See the [SSL Lite \(CyaSSL\)](#) section for more information.

## SSL Overview

Nucleus SSL supports SSL 2.0, SSL 3.0, and TLS 1.0. Unless otherwise noted, this guide uses SSL to refer to both SSL or TLS.

Nucleus SSL is almost completely compatible with the Open Source “OpenSSL” project. Therefore, you should refer to the OpenSSL documentation for a description of all the APIs supported by Nucleus SSL. The “OpenSSL” documentation is located here: <http://www.openssl.org/docs/ssl/ssl.html>.

The only API-level deviation of Nucleus SSL from “OpenSSL” is that all API, data structures, and macro names of “OpenSSL” which contain the string “RC4” or “rc4” should be replaced by “ARC4” or “arc4”, respectively. The following are some examples of the changes:

- The “SSL\_RC4” macro should be renamed to “SSL\_ARC4”.
- The “EVP\_rc4” function name should be renamed to “EVP\_arc4”.

## MMU Interaction Limitation

This version of Nucleus SSL is not compatible with MMU directly; that is, no API can be called directly from the user space. Nevertheless, other middleware products which are using SSL internally can safely use it with MMU also (for example, Nucleus WebServ).

## SSL Negotiation

SSL goes through different stages during an SSL connection. The first stage is the SSL negotiation, during which the following takes place:

- Public keys are exchanged.
- Versions of SSL are decided upon.
- The cipher suite is chosen.
- Certificates are passed.

If for any reason there is a disagreement between the client and server, communications are dropped and the SSL connection ceases.

## Cipher Suites

There is more than one encryption algorithm that can be used for securing data, and more than one algorithm will be used during the course of a session. An in-depth description on how these suites work is discussed later in this guide.

## Certificates

To assure that the secure session is with a valid peer, it is necessary to verify the other connection. This is done through certificate exchange. A certificate is similar to a driver's license in that it has been issued by a third party and can be trusted as a means of identification. Certificates are discussed in further detail later in this guide.

## SSL Session

Once the negotiation phase has been completed and both sides are in agreement, the SSL session begins. This involves the sending and receiving of data through an open TCP socket. All data being sent to the TCP layer of the network protocol stack will be encrypted. Thus, if you are using HTTP, the HTTP data and the header will be encrypted.

## Shutdown

Finally, comes the shutdown phase. One of the two sides should initiate the shutdown phase when all communications cease. This lets both sides know that there will be no more data sent. The client can simply free up any memory associated with the session and then close the socket. The server will save the session information in case the same client may try to activate another session within a specified time. The saved information quickens the SSL negotiation phase since the SSL version and cipher suite have already been agreed upon.

## SSL Functionality

To use SSL with Nucleus, you need to initialize Nucleus NET. At this point, the client and server begin to act a little differently.

An SSL client will open a socket with a server. This socket must bind to the port the server deems as an SSL-specific port. For the Internet, this port number is 443. A server will open a socket within Nucleus NET and bind itself to a specific port. Care should be taken not to bind to the port already in use. The server then waits for a client to make a connection.

Once the client connects, the SSL negotiations can take place. This is done through a set of function calls to Nucleus SSL. When the last function call completes, communications can take place. All sends and receives should now go through Nucleus SSL. Finally, the shutdown process takes place. The memory is freed and the socket is closed. This process is now ready to be repeated.

## Creating an Application

A SSL client initiates a request for a secure communication session. Once the server receives the request, the client informs the server of the types of communications it can perform.

A SSL server responds to requests for secure communications. Once the server receives the request, it decides the parameters of the SSL connection. Once these parameters have been decided, either side can send encrypted data.

This chapter explains how to set up a SSL application. This will begin by looking at initializing the application, followed by connecting to or accepting connections, receiving and sending data, and finally, closing the connection.

## SSL Initialization

How the server communicates with a client is primarily determined by initialization. Many defines can be configured to help customize and reduce the size of the server or client. The main Nucleus SSL configuration file is named *nu\_openssl\_cfg.h*. This file is described in [SSL Configuration File](#).

## Connecting to a Server

For SSL communications to occur, a client must initiate a connection. This connection can be made through Nucleus NET function calls. Once the client has connected to the correct port on the server's device, SSL negotiations can begin. Upon completion of the SSL negotiation, the connection has either been accepted or rejected.

## Nucleus NET Routines

Following the Nucleus NET rules for setting up a connection is required. Here is a brief description of the events that must occur. Examples come from the demonstration provided in *ssld.c*, `SSL_Client_Task`.

The client must first open a socket by which it will connect to the server:

```
client_socket = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);
if(client_socket >= 0)
{
    /* Fill in a structure with the server address. */
    servaddr.family      = NU_FAMILY_IP;
    memcpy(&servaddr.id, SSL_Server_Ip, IP_ADDR_LEN);
    servaddr.port        = SSL_SERVER_PORT;
    servaddr.name        = "SSL_Client";

    if((NU_Connect(client_socket, &servaddr, 0)) >= 0)
    {
```

At this point, the client is connected to the server through port `SSL_SERVER_PORT`, 443. The application must now inform the server that SSL communications are required. This is done by making a call into the SSL library to connect.

```
/* Begin SSL negotiations. */
SSLD_Connect(&ssl_struct, client_socket, NU_SSL_CTX);
```

`SSLD_Connect` initiates the SSL negotiations. Once this function returns, `ssl_struct` will be filled in according to the template structure `NU_SSL_CTX`, and is specific to this SSL session.

The application is now able to perform secure communications.

## Accepting Connections

For SSL communications to occur, a client must initiate a connection. This requires the server to wait for the client to connect to its device. This waiting is done through Nucleus NET function calls. Once the client has connected to the correct port on the server's device, SSL negotiations can begin. Upon completion of the SSL negotiation, the connection has either been accepted or rejected.

## Nucleus NET Routines

Following the Nucleus NET rules for setting up a socket is required. Here is a brief description of the events that must occur. Examples come from the demonstration provided in *ssld.c*, `SSL_Server_Task`.

The server must first open a socket and bind to the port, which will accept incoming connections:

```
socketd = NU_Socket(NU_FAMILY_IP, NU_TYPE_STREAM, 0);
```

```
if(socketd >=0)
{
    /* Fill in a structure with the server address. */
    servaddr.family    = NU_FAMILY_IP;
    servaddr.port      = (UINT16)SSL_SERVER_PORT;
    *(UINT32 *)servaddr.id.is_ip_addrs = IP_ADDR_ANY;
    servaddr.name       = "SSL_Server";

    /* Make an NU_Bind() call to bind the server's address. */
    if ((NU_Bind(socketd, &servaddr, 0)) >= 0)
    {
```

The server listens for data on port `SSL_SERVER_PORT` through the call to `NU_Listen`, and accepts any incoming connections through the call to `NU_Accept`. It should be noted that the accepting state occurs within an infinite loop. This allows the server to accept more than one client.

```
/* Be ready to accept connection requests. */
status = NU_Listen(socketd, 10);

if (status == NU_SUCCESS)
{
    while(1)
    {
        /* Block in NU_Accept until a client connects. */
        new_socket = NU_Accept(socketd, &client_addr, 0);
        if (new_socket >= 0)
        {
```

When a client connects with the server, `NU_Accept` will return the socket descriptor of that new socket created for the session. Client/server communications can now begin. It is up to the application to begin the SSL session.

```
/* Begin SSL negotiations */
SSLD_Accept(&ssl_struct, new_socket, NU_SSL_CTX);
```

`SSLD_Accept` initiates the actual SSL negotiations. Once this function returns, `ssl_struct` will be filled in according to the template structure `NU_SSL_CTX`, and is specific to this SSL session.

The application is now able to perform secure communications.

## Transferring Data

Once the client/server connection has been established, data can be securely transferred. This transfer does not need to be initiated by the client, and must follow very few rules.

The socket the client and server are communicating through is set up for SSL. Because of this, all communications must use the function calls `SSL_read` and `SSL_write`. Direct access to Nucleus NET by using `NU_Recv` and `NU_Send` must not be made. This will disrupt the communications resulting in the other side ceasing the connection.

When making the calls to `SSL_read` and `SSL_write`, the session structure initialized by `SSLD_Accept` or `SSLD_Connect` must be used. This structure keeps track of information necessary for encrypting and decrypting the data being transferred.

## Receiving Data

The following is a continuation of the demonstration application provided within *ssld.c*.

```
if(ssl_struct)
{
    /* Receive the request packet. */
    bytes_received = SSL_read(ssl_struct, rx_buf, 256);

    if(bytes_received > 0)
    {
```

`SSL_read` returns the number of bytes received from the other side. If the result is negative, the value will correspond with a Nucleus NET error code. If data does exist, it will be placed within `rx_buf`, decrypted, and is ready to be processed by the application.

## Sending Data

The following is a continuation of the demonstration application provided within *ssld.c*.

```
while((MAX_SEND_SIZE < size) && (bytes_sent >= 0))
{
    bytes_sent = SSL_write(ssl_struct, tmpptr, MAX_SEND_SIZE);
    size -= MAX_SEND_SIZE;
    tmpptr += MAX_SEND_SIZE;
}

if(size && (bytes_sent >= 0))
{
    bytes_sent = SSL_write(ssl_struct, tmpptr, size);
}
```

To send data, the total size of the data being sent must be known. `SSL_write` returns the number of bytes actually sent to the other side of the connection. If this value is negative, the value will correspond with a Nucleus NET error code.

## Terminating the Session

The session must be terminated when either an error has occurred during communications or communications are finished. When terminating the session, memory will be freed and information will be stored into cache. It is necessary to do the following at the end of each session:

```
/* Close the connection. */
SSL_shutdown(ssl_struct);
SSL_free(ssl_struct);
NU_Close_Socket(new_socket);
```

SSL\_shutdown verifies the other side knows the connection is being closed. As part of the SSL standard, it is required to inform the peer when communications will be terminated. SSL\_free returns memory to the memory pool that was allocated during the session. This amount may vary from session to session. Finally, the socket must be closed through a call to Nucleus NET. Once this is done, the session is completed and the task is free to pursue more SSL communications.

## Real-Time Clock

A typical SSL session includes an exchanging of certificates. This is usually done when a client connects to a server and wants to verify the server is authentic. The server passes its certificates, which contain three important bits of information: host name, certificate authority, and expiration of the certificate.

If the application is going to verify certificates, it must have access to accurate time. This can be achieved by using a real-time clock in the hardware.

The file *crypto\_nucleus.c* needs to be modified for this purpose. The `cos_time()` function in this file should be modified to return the real-time clock time since Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

If a real-time clock is not present, other solutions such as using a network time protocol should be explored. The default implementation in *o\_nucleus.c* hard codes the clock to a particular date/time by presetting the OPENSSL Time global variable. This is a stub implementation which you must replace.

If certificates are not to be verified, as is typical operation within a web server, then a real-time clock is not necessary.

## Authentication vs. Encryption

Nucleus SSL uses both authentication and data encryption to enforce a secure connection. Different algorithms are used for these methods.

## SSL Authentication

Authentication is the process of verifying an entity is who they claim to be. This can be as simple as evaluating a login name and password or verifying an IP address, or as complicated as querying a trusted third party to acknowledge the entity as being valid. This does not, however, require that the data being passed between server and client be secure.

Authentication may be used to allow only privileged users access, or the ability to modify a system, or to verify sensitive data is being exchanged only with a privileged user. Nucleus SSL uses certificates to authenticate the latter circumstance.

## Hashing Algorithms

A hash algorithm defines a function which possibly takes a larger amount of data and, through mathematical computations, reduces it down to a smaller, fixed size. Within SSL, hash functions are used as a signature, helping verify the origination of the document. While the client and server are negotiating a secure session, two types of hashing may be used. These are Message-Digest 5 (MD5) and Secure Hash Algorithm (SHA).

## SSL Encryption

Encryption is the modification of data so that non-privileged entities cannot make sense of it. There are many cryptographic algorithms available to perform encryption, each with its strengths and weaknesses. These algorithms can be divided into two groups: asymmetric, and symmetric. Within SSL, these can also be called key exchange algorithms and data encryption algorithms, respectively.

### Asymmetric Algorithms

An asymmetric algorithm is one where a first key is used for encrypting, and a second key is used for decrypting. Since the encrypting key cannot decrypt, it can be made public, thus it is known as Public Key Encryption. This type of encryption is used during session negotiation. Asymmetric encryption is very taxing on the processor, and would be very inefficient to use during the data encryption phase.

The three forms of asymmetric encryption provided within Nucleus SSL are Diffie-Hellman (DH), Rivest Shamir Adleman (RSA), and Digital Signature Algorithm (DSA).

### Symmetric Algorithms

Symmetric encryption uses the same key for encrypting and decrypting. If this key were made public, any entity would have the ability to decrypt what has been encrypted with this key; therefore, this type of encryption is known as Private Key Encryption.

During the session negotiation, the private key is encrypted using asymmetric encryption, and then sent to the peer. This key is then used during the data encryption phase to reduce the strain on the processor.

The different forms of symmetric encryption supplied with Nucleus SSL are Data Encryption Standard (DES, 3DES), Rivest Cipher (RC2, Arc4), and several others.

## SSL Example

An SSL session can be divided into two distinguishable parts; session negotiation and data transfer. Data transfer cannot happen without a successful completion of session negotiation. This chapter explores a SSL session in more detail than previously described within this guide. It discusses the packet structure, session negotiation communications, and the data transfer.



## SSL Packet

What is referred to in SSL as the Record Layer is the mechanism that encapsulates all data being sent to the lower networking layers. This includes creating the packet header. The SSL packet header is described as in [Table 6-1](#).

**Table 6-1. SSL Packet**

Header Field	Size (bytes)	Description
Protocol	1	The type of packet.
Version	2	Version of SSL being used. TLS 1.0 is described as SSL 3.1.
Length	2	Length of the data field. SSL specifications do not allow the size to exceed 214.
Data	Variable	The data can be supplied by SSL or by an application. The data area is usually encrypted.

SSL currently allows for four different protocol types, as shown in [Table 6-2](#):

**Table 6-2. SSL Protocol Types**

Value	Description
20	ChangeCipherSpec - This is generally used to announce that encryption will now be used.
21	Alert - This is used to communicate an error or caution condition.
22	Handshake - The handshake protocol used during session negotiation and should be considered the heart of SSL.
23	Application Data - This is data passed to SSL by another application.

This packet description is a simplified version of the actual packet description. It is supplied to give the basic understanding of how a client and server communicate. For a more in-depth discussion of the packet, please refer to the SSL Standard or TLS RFC 2246.

## Session Negotiation

Within the SSL Standard and TLS RFC, the session negotiation is given the most emphasis. The session negotiation includes two different encryption negotiations; hashing and key exchange. This is all done through a series of handshakes that are described in this section.

Each of the following steps designates a separate packet transfer. The origination of the message being sent is shown within the parentheses.

### 1. Client Hello (client side)

This signals the beginning of SSL session negotiations. A peer is requesting secure communications with another peer. Within the SSL Handshake packet the client sends a 32 byte random number. If the client is resuming a previous session, the session ID is also passed.

Following this data is a list of potential ciphers that the client can perform. The cipher suites are described by a two-byte code. The cipher codes also take into account the version of SSL being used. Though the compression methods field is not used, it is set up for future use. This last field allows the client to specify different compression methods that may be used during the session.

It is important to note that the client has only conveyed how it can perform the session, and not how the session will be performed. In other words, the client is obliged to give the server multiple choices so that the server can decide which parameters to use.

### 2. Server Hello (server side)

The server hello message is the counterpart to the client hello message. It first passes the 32 byte random number. Following this a session ID may be passed. This is only passed if the server allows continuation of SSL sessions. Allowing this greatly speeds up all subsequent negotiations that the client and server perform.

Lastly, the server specifies the parameters that are to be used during this connection. As the client stated which ciphers it could use, the server now specifies which cipher is used. The cipher that the server chooses must be on the list of ciphers that the client has sent. In the future, compression will be handled this same way.

If there is a disagreement in which cipher to use, the negotiations are canceled. The disagreement occurs when the cipher does not use any of the ciphers that the client has proposed.

### 3. Certificate (server side)

Once the server has sent the server hello message, it then sends its certificates. The certificate chain begins with the sender's certificate. The certificate of the authority that granted the first certificate then follows this certificate. The next certificate, if one is present, is from the authority that granted the previous authority's certificate. This chain can continue for an unspecified number of certificates.

As discussed in [IPsec-IKE](#) chapter, the client uses the certificate to help verify that the server can be trusted.

#### **4. Server Key Exchange (server side)**

Now that the cipher suite has been chosen and the certificate has been passed to the client, the server can send the public key. The client uses this key to encrypt the rest of the negotiations. As shown in the [IPsec-IKE](#) chapter, it is too costly to use a public key algorithm for the entire SSL session. Once the secure negotiations begin, the private key can be exchanged and the secure session can begin.

#### **5. Server Hello Done (server side)**

The server has now finished its hello negotiations. A message is sent to the client to allow the client to respond to the server's session negotiation.

#### **6. Client Key Exchange (client side)**

Now that the client has the server's public key, it can encrypt the private key used for the symmetric encryption algorithm to take place during the SSL session. As is the nature of public key encryption, only the server can decrypt the client's private key. This gives the client more assurance that it is communicating with the correct entity.

This marks the end of the initial SSL negotiation.

#### **7. Change Cipher Spec (client side)**

With this message, the client is notifying the server that it will begin encrypting future messages with the new key. For the client to reach this state, the following must be defined for this session: specific symmetric encryption algorithm, specific message integrity algorithm, and a specific key used for such algorithms. Without this completed list, errors in communications are likely.

#### **8. Finished (client side)**

The client now states that the negotiation was successful and is ready for secure data transmission.

#### **9. Change Cipher Spec (server side)**

As with the client, the server must send a message stating that it is switching encrypted communications to the symmetric algorithm. This can only be done if the server is clear on all algorithms and keys.

#### **10. Finished (server side)**

This message signifies the end of SSL session negotiation. Once the server sends this message, secure data transmission may occur.

## Data Transfer

Once the SSL session has begun, the application may now pass its private data through SSL to get encrypted. Creating the data packet that is sent to the TCP networking layer is done in two parts. First, a hashing algorithm is used on the unencrypted data, and, then, the data and the hash are encrypted.

## Message Authentication

To verify that the message received has not been altered in transit, a message authentication code is appended to the message. The hashing algorithm that was determined within the session negotiation creates this code. There are two algorithms used by SSL: Message Digest 5 (MD5) and Secure Hash Algorithm (SHA).

The main difference between the two algorithms is the size of the end code. MD5 creates a code that is 16-bytes in length, and SHA creates a 20-byte code. When the data is received from the application, the hash algorithm is performed. The data is then appended with the authentication code and encrypted. When the message is received and decrypted, the code can then be used to verify authenticity.

## Encryption

The purpose of SSL is to provide a secure transfer of data through a public medium. After the session has begun, and the authentication code created, encryption can finally happen. The data that is passed to the SSL layer is appended with the message authentication code and encrypted using the key and algorithm decided upon during session negotiation. A successful SSL session has now been achieved.

## SSL Configuration File

The SSL configuration file can be used to configure the Nucleus SSL product. The code and data footprint of Nucleus SSL can be reduced by removing different features of Nucleus SSL, but this may hinder or prevent the ability to communicate through SSL to other applications.

### nu\_openssl\_cfg.h

This configuration file contains a group of macros which are mostly “undefined” by default. Modifying the configuration file to “define” these macros lets you disable individual components of Nucleus SSL which are not needed by the user application. For example, if you want to disable the SHA512 hash algorithm, then you must change the file as follows:

Original line in file:

```
#undef OPENSSSL_NO_SHA512
```

Updated line in file:

```
#define OPENSSSL_NO_SHA512
```

## SSL Lite (CyaSSL)

SSL Lite is used for applications that require secure connections within a small footprint. SSL Lite uses the CyaSSL library to enable secure communication.

---

### Note



SSL Lite cannot be enabled with Nucleus SSL simultaneously.

---

SSL Lite is integrated with Nucleus [HTTP Lite](#) and Nucleus [WebSockets](#) components. The configuration of secure connections can be accomplished through metadata options for the SSL Lite component. See CyaSSL documentation for descriptions of these options. The CyaSSL documentation is available here:

<http://www.yassl.com/yaSSL/Docs.html>









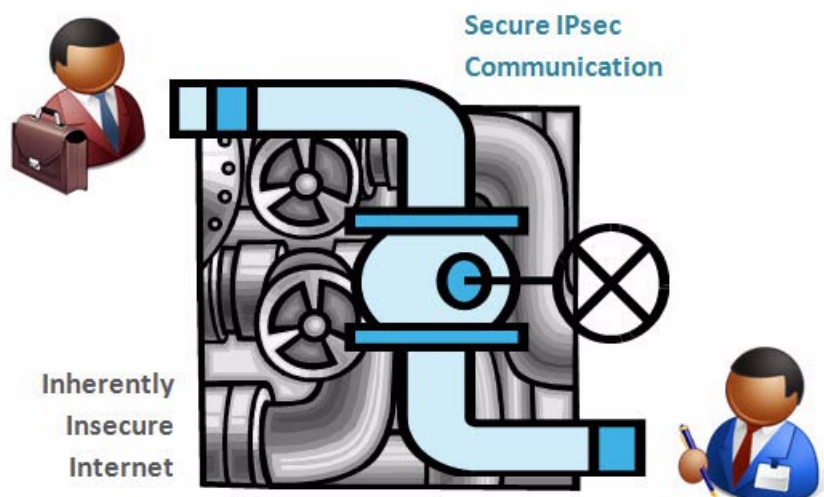




## IPsec Introduction

The Internet provides a cost-effective method of communication. However, IP, the network protocol that drives the Internet is inherently insecure. Many organizations desire to use the Internet Infrastructure for communications between remote locations. Additionally they require protection of their communication channels from unauthorized viewing as well as a need to authenticate the entity at the other side of the channel.

**Figure 7-1. IPsec Communication Example**



---

### Note



This document is not a reference on IPsec. It is only a brief discussion about the IPsec concepts. Readers are directed to the large amounts of literature available on this subject. Suggested reading: “IPsec - The New Security Standard for the Internet, Intranets, and Virtual Private Networks” by Naganand Doraswamy and Dan Harkins (ISBN 0-13-011898-2)

---

IPsec is a standardized protocol that enables security on IP packets. The following services are provided by IPsec:

- Data origin authentication  
Verifies that each packet is delivered by the claimed sender.

- Connection less data integrity authentication  
Ensures that the contents of the network packets are not modified during transmission, either deliberately or non-deliberately.
- Data content confidentiality  
Conceals the contents of network packets by encrypting them.
- Anti-replay protection  
Ensures that network packets cannot be replayed at a later time by a malicious sender.
- Limited traffic flow confidentiality  
Ensures that security cannot be breached just monitoring network traffic.

IPsec uses two protocols to provide security. Either one or both of these protocols can be used to protect any type of IP traffic:

- Encapsulating Security Payload (ESP) is an IPsec protocol which provides all the above listed services.
- Authentication Header (AH) is another IPsec protocol that provides all the above listed services except “data content confidentiality”. In other words, the AH protocol is not capable of encrypting contents of the IP packet.


IPsec enforces security by making use of two types of information. Both of these two pieces are required to establish a secure communication channel between two nodes on a network:

- Security Policy (SP): This defines a policy between two networks or hosts. A policy dictates the encryption and authentication algorithms to use to secure data and the level of security to be applied to the network packets between two entities.
- Security Association (SA): Unlike SP, which is general and only defines security algorithms to be used, SA can be considered an incarnation of a policy. A Security Association contains the actual algorithm “keys” which are used to protect the data. This is the most important piece of information for security because if the keys are compromised, all data secured with IPsec can easily be decrypted.

All IPsec implementations usually contain two databases. The Security Association Database (SAD) contains a list of established Security Associations, whereas the Security Policy Database contains a list of Security Policies.

## Nucleus IPsec

This chapter describes the Internet Protocol Security (IPsec) services and policy definitions supported by Nucleus. It contains the Nucleus software with the specific Encryption and Authentication algorithms supported and provisions in the software for extending support for new Encryption and Authentication algorithms.

**Warning**  In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

## Encapsulating Security Payload (ESP)

Encapsulating Security Payload (ESP) is a protocol header inserted in an IP packet to provide confidentiality, data origin authentication, anti-replay, data integrity, and limited traffic flow confidentiality services.

Confidentiality is provided through an Encryptor and data integrity with an Authenticator algorithm. [Table 7-1](#) lists the encryption and authentication algorithms supported by Nucleus IPsec.

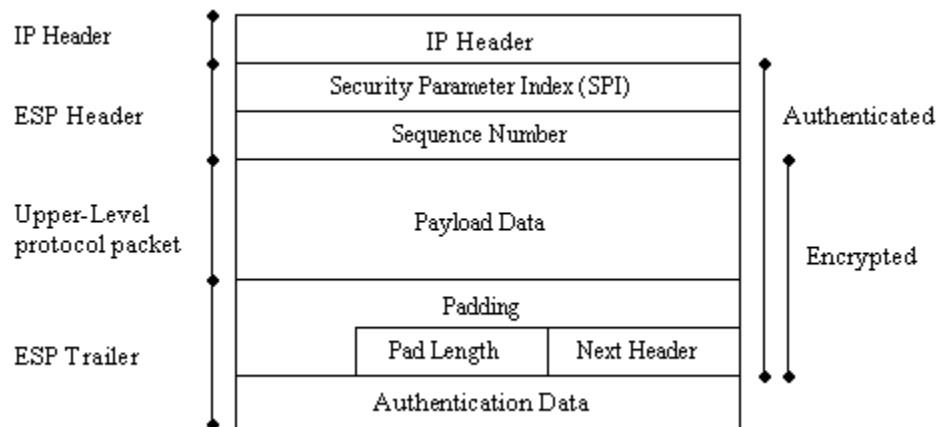
**Table 7-1. Supported IPsec Algorithms**

Encryption Algorithm	Authenticator Algorithm
<ul style="list-style-type: none"><li>• DES-CBC Cipher</li><li>• 3DES-CBC Cipher</li><li>• AES-CBC Cipher</li><li>• BLOWFISH-CBC</li><li>• CAST-128 Encryption Algorithm</li><li>• NULL Encryption Algorithm</li></ul>	<ul style="list-style-type: none"><li>• HMAC-MD5-96</li><li>• HMAC-SHA-1-96</li><li>• HMAC-SHA-256-96</li><li>• AES-XCBC-MAC-96</li><li>• NULL Authentication Algorithm</li></ul>

## ESP Packet

The ESP header always follows an IP header, regardless of the mode in which ESP is being applied as shown in [Figure 7-2](#). If ESP is being applied in Transport Mode, the ESP header is placed between an IP header and a higher-layer protocol header such as TCP. In Tunnel Mode, the ESP header lies between two IP headers.

**Figure 7-2. IP Packet Being Secured Using ESP**



The Encryptor in an ESP protocol encrypts the complete IP payload only in Transport Mode, for example, a TCP header and its payload. While in Tunnel Mode, it encrypts the complete IP packet, for example, the IP header as well as its complete payload. The Padding, Pad Length, and the Next Header fields from the ESP trailer are also encrypted in both modes. [Figure 7-2](#) describes the header fields.

The Authenticator in an ESP protocol calculates a digest on the complete ESP header, higher-layer packet, and ESP trailer other than the Authentication Data field.

## ESP Header

**Table 7-2. ESP Header**

Fields	Description
Security Parameter Index (SPI)	This value, combined with the destination address and the upper-layer protocol, identifies the Security Association to be used.
Sequence Number	Counter that provides anti-replay service.

## ESP Trailer

**Table 7-3. ESP Trailer**

Fields	Description
Padding	Padding is used to maintain aligned boundaries. Some cipher algorithms require that the input be a multiple of its block size. Padding is used to accommodate this.
Pad Length	Length of the padding used.
Next Header	The type of data contained in the payload data field. When ESP is applied in Tunnel Mode, this field is set to IP_IN_IP for IPv4 and IP6_IN_IP6 for IPv6. When applied in Transport Mode, this would be set to the higher-layer protocol.
Authentication Data	The Authentication Digest calculated on the complete ESP header, higher-layer packet and ESP trailer other than the Authentication Data field.

## Authentication Header

The Authentication Header (AH) is a protocol header inserted in an IP packet to provide data origin authentication, anti-replay, and data integrity services. AH does not provide confidentiality.

Authentication is provided through an Authenticator algorithm. The following is a list of the authentication algorithms supported by Nucleus IPsec for AH.

## Supported Authentication Algorithms

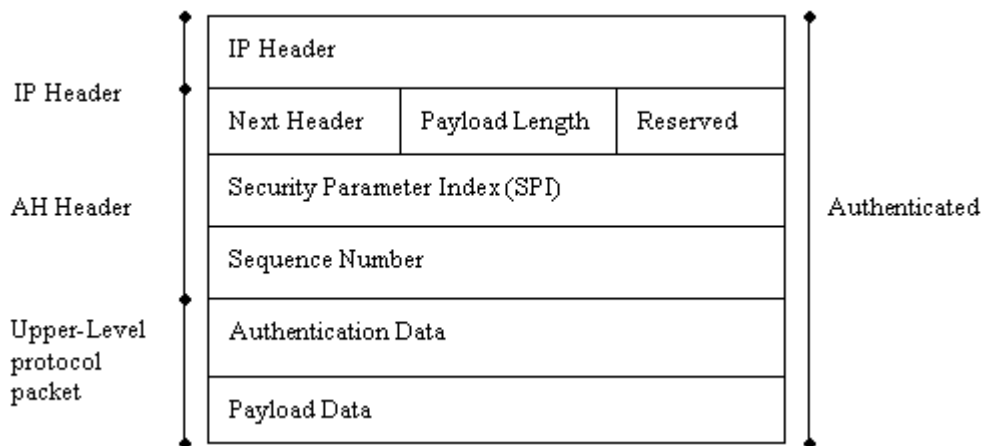
- HMAC-MD5-96
- HMAC-SHA-1-96
- HMAC-SHA-256-96

- AES-XCBC-MAC-96

## AH Packet

Like the ESP header, an AH header always follows an IP header, regardless of the mode in which AH is being applied. If AH is being applied in Transport Mode, the AH header is placed between an IP header and a higher-layer protocol header such as TCP. In Tunnel Mode, the AH header lies between two IP headers.

**Figure 7-3. IP Packet Being Secured Using AH**



The Authenticator in an AH calculates a digest on the IP header, AH header and higher-layer packet. Note that the Authentication Data field is within the area being authenticated. When calculating the digest, this field is zeroed. The value of the digest is then placed here.

## AH Header

The AH header is described in [Table 7-4](#):

**Table 7-4. AH Header**

Fields	Description
Next Header	The type of data contained in the payload data field. When AH is applied in Tunnel Mode, this field is set to IP-IN-IP for IPv4 and IP6_IN_IP6 for IPv6. When applied in Transport Mode, this is set to the higher-layer protocol.
Payload Length	Length of the header in 32-bit words minus two.
Reserved	This field is reserved.
Security Parameter Index (SPI)	This value, combined with the destination address and the upper-level protocol, identifies the Security Association to be used.



**Table 7-4. AH Header (cont.)**

Fields	Description
Sequence Number	Counter that provides anti-replay service.
Authentication Data	The authentication digest calculated on the IP header, AH header, and the upper-layer packet.

## Encryptor

The Encryptor component is invoked to encrypt a given packet of data. This component interfaces with the Nucleus SSL crypto library to provide encryption algorithms to the ESP component of Nucleus IPsec.

## Encryption Algorithms Supported

Nucleus IPsec supports the following Encryption algorithms to be used with ESP:

- DES-CBC Cipher
- 3DES-CBC Cipher
- AES-CBC Cipher
- BLOWFISH-CBC
- CAST-128 Encryption Algorithm
- NULL Encryption Algorithm

## IPsec Configurations

The following is a description of the macros that define whether each of these algorithms will be included in the build. The macros are defined in *ipsec/ips\_cfg.h*.

---

**Note**

IPsec conformance requires support for DES-CBC and NULL Encryption algorithms.

---

### IPSEC\_INCLUDE\_DES

By default, this macro is set to NU\_TRUE, and Nucleus IPsec is compiled with DES-CBC support. Set this macro to NU\_FALSE to exclude the algorithm from the build.

### IPSEC\_INCLUDE\_3DES

By default, this macro is set to NU\_TRUE, and Nucleus IPsec is compiled with 3DES-CBC support. Set this macro to NU\_FALSE to exclude the algorithm from the build.

## IPSEC\_INCLUDE\_AES

By default, this macro is set to `NU_FALSE`, and Nucleus IPsec is compiled without AES-CBC support. Set this macro to `NU_TRUE` to include the algorithm in the build.

## IPSEC\_INCLUDE\_BLOWFISH

By default, this macro is set to `NU_TRUE`, and Nucleus IPsec is compiled with BLOWFISH-CBC support. Set this macro to `NU_FALSE` to exclude the algorithm from the build.

## IPSEC\_INCLUDE\_CAST128

By default, this macro is set to `NU_TRUE`, and Nucleus IPsec is compiled with CAST-128 support. Set this macro to `NU_FALSE` to exclude the algorithm from the build.

## IPSEC\_INCLUDE\_NULL\_ENCRYPTION

By default, this macro is set to `NU_TRUE`, and Nucleus IPsec is compiled with NULL Encryption support. Set this macro to `NU_FALSE` to exclude the algorithm from the build.

---

### Note



Nucleus IPsec supports transparently adding support for new authentication algorithms. Refer to the "[Customizations](#)" on page 1269" of this guide for more information.

---

# Security Databases

This section provides you with some core definitions related to security databases.

## Security Association

A Security Association (SA) is an agreement between two entities on how IPsec will be applied to a particular packet. Each host has two SAs for each target host with which it is communicating, one for inbound traffic and another for outbound traffic. The inbound SA is used by the host to process incoming IP packets and the outbound SA is used to process outgoing packets.

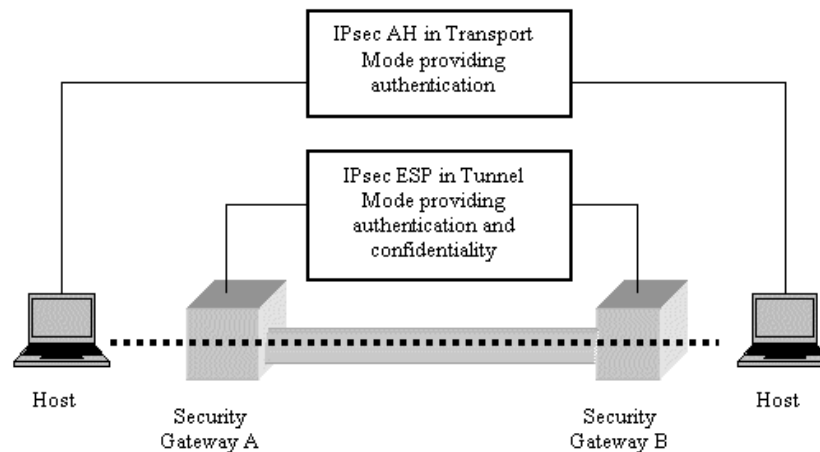
## Security Policy and Security Policy Database

A security policy is used to determine how to process a particular IP packet, and it contains the necessary information for any required action. A security policy can belong to only one SPD and a Security Policy Database (SPD) can belong to only one IPsec group. The following [Figure 7-4](#) shows this hierarchy. The SPD is consulted for both inbound and outbound traffic. Refer to "[Nucleus IPsec Processing](#)" on page 1271 for details on inbound and outbound processing.

## Security Policy Group

Interfaces are associated with a Security Policy Group, and all packets received on an interface are processed according to the group with which the interface is associated. Each Security Policy Group has its own Security Association Database (SADB) and SPDB.

**Figure 7-4. Nucleus IPsec Database**



## Security Policy

Nucleus IPsec maintains a list of security policies for each group. Both outbound and inbound traffic is processed by the networking stack only if an IPsec policy exists in the SPDB of a security group which allows this. For example, if there is only one policy that allows TCP traffic over port 23, no other traffic is allowed to pass through except to port 23. If, for example, you also want to allow UDP traffic, another policy must be added to the SPDB.

Each policy in the SPDB of a security group is uniquely identified by an unsigned 32-bit integer which can be used at the application level to identify the policies.

## Policy Sorting Order

The policy sorting order plays a very important role in the IPsec protocol. When matching incoming or outgoing packets against policies, the search starts from the first policy in the list and terminates on the first match found. A packet may match both a range-based and a single IP-based IPsec policy, but only the first one found is used.


Under default configuration, Nucleus IPsec uses a simple approach to ordering policies. Policies added first are stored first in the policy list.

## IPsec Data Structures

This section describes the Nucleus IPsec-specific data structures used by the various API level routines. A description of each data structure and its corresponding members is provided.

---


**Note**

 Some members in the structures may have been omitted, as they are used internally by Nucleus IPsec.

---

---

**Note**

 All structures have read-write access unless otherwise stated.

---

### IPSEC\_POLICY

The IPSEC\_POLICY is the typedef for the ipsec\_policy data structure used to define policies.

```
typedef struct ipsec_policy
{
    #if ( IPSEC_ENABLE_PRIORITY == NU_TRUE )
        UINT32                ipsec_priority;
    #endif
    IPSEC_SECURITY_PROTOCOL    *ipsec_security;
    IPSEC_SELECTOR             ipsec_select;
    IPSEC_BUNDLE_LIST          ipsec_bundles;
    #if (INCLUDE_IKE == NU_TRUE)
        IPSEC_SA_LIFETIME      ipsec_sa_max_lifetime;
        UINT16                 ipsec_pfs_group_desc;
    #endif
    UINT8                      ipsec_security_size;
    UINT8                      ipsec_flags;
} IPSEC_POLICY;
```

The members of this structure are defined in [Table 7-5](#). Some members of this structure have been omitted as they are used internally:

**Table 7-5. IPSEC\_POLICY**

Member	Description
ipsec_priority	Indicates the priority of a policy. Valid priorities range from 0 to 4294967295. 0 is the highest priority and 4294967295 is the lowest priority. Policies with the same priority are sorted according to the order they were entered in to the policy database (those policies that are entered first have a higher order). IPSEC_ENABLE_PRIORITY must be set to NU_TRUE to use this feature.

Table 7-5. IPSEC\_POLICY (cont.)

Member	Description
ipsec_security	( <a href="#">IPSEC_SECURITY_PROTOCOL</a> *) An array indicating how security should be applied to packets matching this policy. Refer to the following description of the <a href="#">IPSEC_SECURITY_PROTOCOL</a> data structure. Note that when a packet is sent, the security is applied on the packet based on this array of security protocols, starting from the first element to the last one. Similarly, when a packet is received, it is decoded with this array of security protocols. However, decoding is in the opposite direction, for example, starting from the last element to the first element in the array.
ipsec_select	The policy's selector used for matching with packet addresses to find a policy in the SPDB. Refer to the description of the " <a href="#">IPSEC_SELECTOR</a> " data structure.
ipsec_bundles	( <a href="#">IPSEC_BUNDLE_LIST</a> ) The bundle list containing all the SA bundles associated with this policy. In IPsec terminology, a set of Security Associations that are applied to one packet is called an SA bundle. A bundle is one or a combination of more than one SA. If a bundle consists of one SA, then only one associated security is applied, if two, then two securities are applied and so on. Note that these SA bundles are handled internally by Nucleus IPsec, and you do not need to know the details for handling them. The only input required from you is the lifetime value. The description of <a href="#">IPSEC_BUNDLE_LIST</a> is provided in the <a href="#">IPsec Data Structures</a> section.
ipsec_sa_max_lifetime	( <a href="#">IPSEC_SA_LIFETIME</a> ) Maximum lifetime for SAs negotiated under this policy. This is only required if IKE support is enabled. By default, it is enabled. Refer to the description of the <a href="#">IPSEC_SA_LIFETIME</a> data structure.

**Table 7-5. IPSEC\_POLICY (cont.)**

Member	Description
ipsec_pfs_group_desc	<p>The IKE group to be used for Perfect Forward Secrecy (PFS). This group defines which Diffie-Hellman group to use during IKE negotiations. This is only required if IKE support is enabled. The following are valid options. If PFS is not required, then no group should be specified, for example, the first option.</p> <ul style="list-style-type: none"> <li>• IKE_GROUP_NONE</li> <li>• IKE_GROUP_MODP_768</li> <li>• IKE_GROUP_MODP_1024</li> <li>• IKE_GROUP_MODP_1536</li> <li>• IKE_GROUP_MODP_2048</li> <li>• IKE_GROUP_MODP_3072</li> <li>• IKE_GROUP_MODP_4096</li> <li>• IKE_GROUP_MODP_6144</li> <li>• IKE_GROUP_MODP_8192</li> </ul> <p>Groups larger than 1024-bits need to be enabled in <i>os/include/networking/nu_networking.h</i>. See the <a href="#">IPsec-IKE</a> chapter for more information.</p>
ipsec_security_size	<p>The number of elements in the ipsec_security array. Note that this value must be less than or equal to the configurable macro IPSEC_MAX_SA_BUNDLE_SIZE, defined in <i>os/include/networking/nu_networking.h</i>.</p>
ipsec_flags	<p>A two-part flag value that specifies the action to be taken for packets matching this policy and the direction of this policy. Valid values to indicate the action to be taken for packets matching this policy are:</p> <ul style="list-style-type: none"> <li>• IPSEC_DISCARD – Packet must be discarded.</li> <li>• IPSEC_BY_PASS – Packet must be bypassed. No security is applied.</li> <li>• IPSEC_APPLY – Packet must be processed with the specified security application.</li> </ul> <p>Only one of these action flags must be set.</p> <p>Valid values to indicate the direction of this policy are:</p> <ul style="list-style-type: none"> <li>• IPSEC_INBOUND – Policy valid for incoming packets only. The selectors are interpreted for an incoming packet.</li> <li>• IPSEC_OUTBOUND – Policy valid for outgoing packets only. The selectors are interpreted for an outbound packet.</li> <li>• IPSEC_DUAL_ASYNCHRONOUS - Single policy for both outbound and inbound packets.</li> </ul> <p>Only one of these direction flags must be set.</p> <p>The direction, action, and selector values of the policy cannot be changed after a policy has been created.</p>

## Related Topics

[IPsec Data Structures](#)[IPSEC\\_Add\\_Policy](#)

# IPSEC\_SECURITY\_PROTOCOL

IPSEC\_SECURITY\_PROTOCOL is the typedef for the ipsec\_security\_protocol data structure:

```
typedef struct ipsec_security_protocol
{
    UINT8          ipsec_tunnel_destination[MAX_ADDRESS_SIZE];
    UINT8          ipsec_tunnel_source[MAX_ADDRESS_SIZE];
    UINT8          ipsec_sa_derivation;
    UINT16         ipsec_security_mode;
    UINT8          ipsec_auth_algo;
    UINT8          ipsec_encryption_algo;
    UINT8          ipsec_protocol;
    UINT8          ipsec_flags;
} IPSEC_SECURITY_PROTOCOL;
```

The members of this structure are described in [Table 7-6](#). Some members of this structure have been omitted as they are used internally.

**Table 7-6. IPSEC\_SECURITY\_PROTOCOL**

Member	Description
ipsec_tunnel_destination	When applying IPsec in Tunnel Mode, this value indicates the tunnel end-point IP address. This could be either IPv4 or IPv6 family type address. Note: The tunnel destination and source of a security protocol must be of the same IP family.
ipsec_tunnel_source	When applying IPsec in Tunnel Mode, this value indicates the tunnel source IP address. This could be either an IPv4 or IPv6 family type address. Note: The tunnel destination and source of a security protocol must be of the same IP family.
ipsec_sa_derivation	Indicates whether the Selector used to search for an SA is to be created from the packet contents, or the Selector from the policy is to be used. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_VALUE_FROM_PACKET</li><li>• IPSEC_VALUE_FROM_POLICY</li></ul> This member is used only when the structure is part of a policy. It is not applicable in the case of Security Associations.
ipsec_security_mode	The mode in which the protocol is applied. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_TRANSPORT_MODE</li><li>• IPSEC_TUNNEL_MODE</li></ul>

**Table 7-6. IPSEC\_SECURITY\_PROTOCOL (cont.)**

Member	Description
ipsec_auth_algo	The algorithm to be used by the Authenticator. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_HMAC_MD5_96</li><li>• IPSEC_HMAC_SHA_1_96</li><li>• IPSEC_HMAC_SHA_256_96</li><li>• IPSEC_NULL_AUTH</li></ul> Note that this field is only required to be set if the AH protocol is specified in the ipsec_protocol field.
ipsec_encryption_algo	The algorithm to be used by the Encryptor. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_DES_CBC</li><li>• IPSEC_3DES_CBC</li><li>• IPSEC_CAST_128</li><li>• IPSEC_BLOWFISH_CBC</li><li>• IPSEC_AES_CBC</li><li>• IPSEC_NULL_CIPHER</li></ul> Note that this field is only required to be set if the ESP protocol is specified in the ipsec_protocol field.
ipsec_protocol	IPsec protocol that is applied. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_AH - Authentication Header.</li><li>• IPSEC_ESP - Encapsulating Security Payload</li></ul>
ipsec_flags	IPsec flags indicating the version of the IP address for the tunnel source and destination. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_IPV4</li><li>• IPSEC_IPV6</li></ul> The following flag can also be set to indicate to Nucleus IKE that this policy requests support for Extended Sequence Number (ESN): <ul style="list-style-type: none"><li>• IPSEC_ENABLE_ESN</li></ul>

## Related Topics

[IPsec Data Structures](#)

[IPSEC\\_Get\\_Policy\\_Opt](#)

[IPSEC\\_Set\\_Policy\\_Opt](#)

## IPSEC\_SELECTOR

IPSEC\_SELECTOR data structure is the typedef for the ipsec\_selector data structure.

```
typedef struct ipsec_selector
{
```



```
union
{
    UINT16      ipsec_src_port;
}ipsec_src_tid;

union
{
    UINT16      ipsec_dst_port;
}ipsec_dst_tid;

IPSEC_IP_ADDR  ipsec_source_ip;
IPSEC_IP_ADDR  ipsec_dest_ip;

UINT8          ipsec_transport_protocol;
UINT8          ipsec_source_type;
UINT8          ipsec_dest_type;
}IPSEC_SELECTOR;
```

The members of this structure are defined in [Table 7-7](#). Some members of this structure have been omitted as they are used internally.

**Table 7-7. IPSEC\_SELECTOR**

Member	Description
ipsec_src_tid.ipsec_src_port	The source port for the IP packets. For example, L2TP uses the port 1701. A wildcard (IPSEC_WILDCARD) may also be used to match all ports.
ipsec_dst_tid.ipsec_dst_port	The destination port for IP packets. A wildcard (IPSEC_WILDCARD) may also be used to match all ports.
ipsec_source_ip	The source address. Refer to the following description of the <a href="#">IPSEC_IP_ADDR</a> data structure. Allowable address classifications are provided.
ipsec_dest_ip	The destination address. Refer to the following description of the <a href="#">IPSEC_IP_ADDR</a> data structure. Allowable address classifications are provided.
ipsec_transport_protocol	The upper-layer protocol. This value is read-only and can only be set when a new policy or SA is being created. Valid values are: <ul style="list-style-type: none"><li>• IP_TCP_PROT</li><li>• IP_UDP_PROT</li><li>• IP_ICMP_PROT</li><li>• IP_ICMP6_PROT</li><li>• IPSEC_WILDCARD</li></ul> A wildcard (IPSEC_WILDCARD) value is used to match all transport protocols.

**Table 7-7. IPSEC\_SELECTOR (cont.)**

Member	Description
ipsec_source_type	<p>The type and family for the source IP address contained in ipsec_source_ip field of the selector. Valid values are:</p> <ul style="list-style-type: none"> <li>• IPSEC_SINGLE_IP - ipsec_source_ip.ipsec_addr contains the single IP address.</li> <li>• IPSEC_RANGE_IP - A range of addresses from ipsec_source_ip.ipsec_addr to ipsec_source_ip.ipsec_ext_addr.ipsec_addr2, inclusive.</li> <li>• IPSEC_SUBNET_IP - ipsec_source_ip.ipsec_addr contains an address and ipsec_source_ip.ipsec_ext_addr.ipsec_addr2 contains the subnet mask. In case of IPv6 address type, ipsec_source_ip.ipsec_ext_addr.ipsec_prefix_len contains a single byte prefix length. The prefix length must be between 1 and 128, inclusive.</li> </ul> <p>Additionally, one of the following flags must also be set to indicate the family type of the IP address:</p> <ul style="list-style-type: none"> <li>• IPSEC_IPV4 or</li> <li>• IPSEC_IPV6.</li> </ul> <p>A wildcard (IPSEC_WILDCARD) may also be used in which case no other flag is set. In this case no IP address or its type will be matched. It results in a successful match.</p>
ipsec_dest_type	<p>The type and family for the destination IP address contained in ipsec_dest_ip field of the selector. Valid values are:</p> <ul style="list-style-type: none"> <li>• IPSEC_SINGLE_IP - ipsec_dest_ip.ipsec_addr contains the single IP address.</li> <li>• IPSEC_RANGE_IP – A range of addresses from ipsec_dest_ip.ipsec_addr to ipsec_dest_ip.ipsec_ext_addr.ipsec_addr2, inclusive.</li> <li>• IPSEC_SUBNET_IP - ipsec_dest_ip.ipsec_addr contains an address and ipsec_dest_ip.ipsec_ext_addr.ipsec_addr2 contains the subnet mask. In case of IPv6 address type, ipsec_dest_ip.ipsec_ext_addr.ipsec_prefix_len contains a single byte prefix length. The prefix length must be between 1 and 128, inclusive.</li> </ul> <p>Additionally, one of the following flags must also be specified to indicate the family type of the IP address:</p> <ul style="list-style-type: none"> <li>• IPSEC_IPV4 or</li> <li>• IPSEC_IPV6.</li> </ul> <p>A wildcard (IPSEC_WILDCARD) may also be used in which case no other flag is set. In this case all IP addresses match the wildcard.</p>

## Related Topics

[IPsec Data Structures](#)[IPSEC\\_Get\\_Inbound\\_SA\\_Opt](#)[IPSEC\\_Get\\_Outbound\\_SA\\_Opt](#)[IPSEC\\_Get\\_Policy\\_Index](#)[IPSEC\\_Get\\_Policy\\_Opt](#)

## IPSEC\_IP\_ADDR

The following is a description of the IPSEC\_IP\_ADDR data structure:

```
typedef struct ipsec_ip_addr
{
    UINT8    ipsec_addr[MAX_ADDRESS_SIZE];
    union
    {
        UINT8    ipsec_addr2[MAX_ADDRESS_SIZE];
        UINT8    ipsec_prefix_len;
    } ipsec_ext_addr;
} IPSEC_IP_ADDR;
```

The members of this structure are defined in [Table 7-8](#).

**Table 7-8. IPSEC\_IP\_ADDR**

Member	Description
ipsec_addr	This can contain a single IP address, the first address of an IP range, or a subnet address. The size of the array is indicated by MAX_ADDRESS_SIZE.
ipsec_ext_addr	<p>A union which is defined as follows:</p> <pre>union {     UINT8 ipsec_addr2[MAX_ADDRESS_SIZE];     UINT8 ipsec_prefix_len; } ipsec_ext_addr;</pre> <p>ipsec_addr2 specifies the last address of an IP range or an IPv4 subnet mask. ipsec_prefix_len specifies an IPv6 prefix length when using IPv6 subnets. The prefix length must be between 1 and 128, inclusive.</p>

## Related Topics

[IPsec Data Structures](#)

## IPSEC\_BUNDLE\_LIST

IPSEC\_BUNDLE\_LIST is the typedef for the ipsec\_bundle\_list data structure:

```
typedef struct ipsec_bundle_list
{
    #if (INCLUDE_IKE == NU_TRUE)
        UINT32      ipsec_lifetime
    #endif
} IPSEC_BUNDLE_LIST;
```

The members of this structure are described in [Table 7-9](#). Some members of this structure have been omitted as they are used internally.

**Table 7-9. IPSEC\_BUNDLE\_LIST**

Member	Description
ipsec_lifetime	The lifetime of the bundle in seconds. All the bundles created within a policy have this common lifetime value. When the timer expires for a bundle, it is removed from the policy's bundle list and is deallocated. If required, a new bundle is created by Nucleus IPsec internally with the same lifetime value. The valid range is from 1 to 4,294,967,295. Note: This is only applicable if IKE is enabled.

## Related Topics

[IPsec Data Structures](#)

## IPSEC\_SA\_LIFETIME

IPSEC\_SA\_LIFETIME is the typedef for the ipsec\_sa\_lifetime data structure. Note that this is only applicable if IKE is enabled.

```
typedef struct ipsec_sa_lifetime
{
    UINT32      ipsec_no_of_secs;
    UINT8      ipsec_expiry_action;
} IPSEC_SA_LIFETIME;
```

The members of this structure are defined in [Table 7-10](#). Some members of this structure have been omitted as they are used internally.

**Table 7-10. IPSEC\_SA\_LIFETIME**

Member	Description
ipsec_no_of_secs	Number of seconds before this timer expires. Valid range is from 1 to 4,294,967,295.

**Table 7-10. IPSEC\_SA\_LIFETIME (cont.)**

Member	Description
ipsec_expiry_action	(UINT8) The action taken when this timer expires. Valid values are: <ul style="list-style-type: none"><li>• IPSEC_LOG_WARNING - Logs a warning.</li><li>• IPSEC_REFRESH_SA - Negotiates a new SA when the old one expires, using IKE. An SA is refreshed only if it has been used at least once in its lifetime and has not explicitly been deleted by the remote host. Refer to the "<a href="#">IPsec SA Re-keying</a>" section in the "Extended Discussion" section of the <a href="#">IPsec-IKE</a> chapter.</li></ul> A warning can be logged along with another action. It is invalid however, to have any other combination of actions.

## IPSEC\_OUTBOUND\_SA

Nucleus IPsec maintains a list of outbound SAs for each policy group. This list is sorted in the same manner as policies are sorted.

The IPSEC\_OUTBOUND\_SA is the typedef for the ipsec\_outbound\_sa data structure used to define an outbound SA. The following is the description of this structure. The values in an SA cannot be changed once added. To change the SA values, the SA must be removed and a new SA added with the new parameters.

```
typedef struct ipsec_outbound_sa
{
    IPSEC_SELECTOR          ipsec_select;
    IPSEC_SECURITY_PROTOCOL ipsec_security;
    UINT32                  ipsec_remote_spi;
    UINT8                   *ipsec_auth_key;
    UINT8                   *ipsec_encryption_key;
} IPSEC_OUTBOUND_SA;
```

The members of this structure are defined in [Table 7-11](#). Some members of this structure have been omitted as they are used internally.

**Table 7-11. IPSEC\_OUTBOUND\_SA**

Member	Description
ipsec_select	The selector for this SA. See the description of the <a href="#">IPSEC_SELECTOR</a> data structure.
ipsec_security	Indicates how IPsec should be applied to packets matching this SA. See the description of the <a href="#">IPSEC_SECURITY_PROTOCOL</a> data structure.
ipsec_remote_spi	Security Parameter Index assigned to this SA by the node with which this association is established. Note that this SA would be an inbound SA for that node.

**Table 7-11. IPSEC\_OUTBOUND\_SA (cont.)**

Member	Description
ipsec_auth_key	<p>Authentication algorithm's key. Note that the length of the key passed should equal the desired algorithm default key size. Failure to provide the key of proper length results in undesired behavior.</p> <p>The following list summarizes the key lengths accepted by supported algorithms and the default key lengths:</p> <ul style="list-style-type: none"> <li>• 0 bytes for NULL authentication</li> <li>• 16 bytes for MD5</li> <li>• 20 bytes for SHA1</li> <li>• 32 bytes for SHA256</li> </ul>
ipsec_encryption_key	<p>Encryption algorithm's key. Note that the length of the key passed should equal the desired algorithm default key size. Failure to provide the key of proper length results in undesired behavior.</p> <p>The following list summarizes the key lengths accepted by supported algorithms and the default key lengths:</p> <ul style="list-style-type: none"> <li>• 0 bytes for NULL cipher</li> <li>• 8 bytes for DES</li> <li>• 24 bytes for 3DES</li> <li>• 16 bytes for AES</li> <li>• 16 bytes for BLOWFISH</li> <li>• 16 bytes for CAST-128</li> </ul>

## Related Topics

[IPsec Data Structures](#)

[IPSEC\\_Add\\_Outbound\\_SA](#)

[IPSEC\\_Add\\_Outbound\\_ESN\\_SA](#)

## IPSEC\_OUTBOUND\_INDEX

IPSEC\_OUTBOUND\_INDEX is the type for the ipsec\_outbound\_index data structure:

```
typedef struct ipsec_outbound_index
{
    CHAR    *ipsec_group;
    UINT32  ipsec_index;
} IPSEC_OUTBOUND_INDEX;
```

The members of this structure are described in [Table 7-12](#).

**Table 7-12. IPSEC\_OUTBOUND\_INDEX**

Members	Description
ipsec_group	A pointer to the name of the group with which the SA is associated.

**Table 7-12. IPSEC\_OUTBOUND\_INDEX (cont.)**

Members	Description
ipsec_index	The index used to uniquely identify the SA in this group.

## Related Topics

[IPsec Data Structures](#)[IPSEC\\_Get\\_Outbound\\_SA\\_Opt](#)[IPSEC\\_Remove\\_Outbound\\_SA](#)

## IPSEC\_INBOUND\_SA

Nucleus IPsec maintains a list of inbound SAs for each group. This list is sorted in ascending order of (SPI, Destination Address, and Security Protocol).

The IPSEC\_INBOUND\_SA is the typedef for the ipsec\_inbound\_sa data structure used to define an inbound SA. The following is the description of this structure. The values in an SA cannot be changed once added. To change the SA values, the SA must be removed and a new SA added with the new parameters.

```
typedef struct ipsec_inbound_sa
{
    IPSEC_SECURITY_PROTOCOL ipsec_security;
    UINT32 ipsec_spi;
    IPSEC_SELECTOR ipsec_select;
    UINT8 *ipsec_auth_key;
    UINT8 *ipsec_encryption_key;
} IPSEC_INBOUND_SA;
```

The members of this structure are defined in [Table 7-13](#). Some of the members of this structure have been omitted as they are used internally.

**Table 7-13. IPSEC\_INBOUND\_SA**

Members	Description
ipsec_security	The protocol being used (AH or ESP) and the mode in which it is being applied. This structure also specifies the Authenticator and Encryptor algorithms that will be used. Refer to the description of the <a href="#">IPSEC_SECURITY_PROTOCOL</a> data structure.

**Table 7-13. IPSEC\_INBOUND\_SA (cont.)**

Members	Description
ipsec_spi	Security Parameter Index of this SA. Note that this SA is an outbound SA from the remote node's perspective. If this value is set to zero and the IPSEC_Add_Inbound_SA() routine is called, then the SPI is assigned to the inbound SA by Nucleus IPsec, and the assigned value returned to the caller. If the routine is called with some non-zero value, then it is assigned to the inbound SA as it is (but it must lie between 256 and 10000).
ipsec_select	The SA selector. Refer to the description of the <a href="#">IPSEC_SELECTOR</a> data structure.
ipsec_auth_key	Authentication algorithm's key.
ipsec_encryption_key	Encryption algorithm's key.

## Related Topics

[IPsec Data Structures](#)

[IPSEC\\_Add\\_Inbound\\_SA](#)  
[IPSEC\\_Add\\_Inbound\\_ESN\\_SA](#)

## IPSEC INBOUND INDEX

The IPSEC\_INBOUND\_INDEX is the typedef for the ipsec\_inbound\_index data structure. Some members may have been omitted as they are used internally.

```
typedef struct ipsec_inbound_index
{
    UINT32    ipsec_spi;
    UINT8     ipsec_dest[MAX_ADDRESS_SIZE];
    UINT8     ipsec_dest_type;
    UINT8     ipsec_protocol;
    UINT8     ipsec_pad[2];
} IPSEC_INBOUND_INDEX;
```

The members of this structure are described in [Table 7-14](#). Some of the members of this structure have been omitted as they are used internally.



**Table 7-14. IPSEC\_INBOUND\_INDEX**

Members	Description
ipsec_dest_type	(UINT8) The type of the Destination Address. Valid values are IPSEC_IPV4 or IPSEC_IPV6.
ipsec_dest	(UINT8 *) An array of size MAX_ADDRESS_SIZE containing the Destination Address of the SA.
ipsec_protocol	(UINT8) The IPsec protocol that is applied. Valid values are IPSEC_AH (Authentication Header) and IPSEC_ESP (Encapsulating Security Payload).
ipsec_spi	(UINT32) The Security Parameter Index.

## Related Topics

[IPsec Data Structures](#)

[IPSEC\\_Get\\_Inbound\\_SA\\_Opt](#)

[IPSEC\\_Remove\\_Inbound\\_SA](#)

## IPsec API Functions

This section discusses the APIs provided for the creation of groups, policies, and security associations.



---

**Note**

To access the APIs and data structures of the IPSEC component, include the file *os/include/networking/nu\_networking.h* in the application.

---

- [IPSEC\\_Add\\_Group](#)
- [IPSEC\\_Add\\_Inbound\\_SA](#) [IPSEC\\_Add\\_Inbound\\_ESN\\_SA](#)
- [IPSEC\\_Add\\_Outbound\\_SA](#) [IPSEC\\_Add\\_Outbound\\_ESN\\_SA](#)
- [IPSEC\\_Add\\_Policy](#)
- [IPSEC\\_Add\\_To\\_Group](#)
- [IPSEC\\_Apply\\_To\\_Interface](#)
- [IPSEC\\_Get\\_Group](#)
- [IPSEC\\_Get\\_Group\\_Opt](#)
- [IPSEC\\_Get\\_Inbound\\_SA\\_Opt](#)
- [IPSEC\\_Get\\_Outbound\\_SA\\_Opt](#)
- [IPSEC\\_Get\\_Policy\\_Index](#)
- [IPSEC\\_Get\\_Policy\\_Opt](#)
- [IPSEC\\_Remove\\_From\\_Group](#)
- [IPSEC\\_Remove\\_Group](#)
- [IPSEC\\_Remove\\_Inbound\\_SA](#)
- [IPSEC\\_Remove\\_Outbound\\_SA](#)
- [IPSEC\\_Remove\\_Policy](#)
- [IPSEC\\_Set\\_Group\\_Opt](#)
- [IPSEC\\_Set\\_Policy\\_Opt](#)

## IPSEC\_Add\_Group

This function adds a new Nucleus IPsec group to the global database of Nucleus IPsec groups. By default, the DF bit processing option is set to clear for every new group. To set different options for the DF bit of a newly created group, refer to the [IPSEC\\_Set\\_Group\\_Opt](#) routine.

### Usage

```
STATUS IPSEC_Add_Group (CHAR *group_name);
```

### Arguments

- `group_name`  
A pointer to the user-defined group name to be added.

### Return Values

- `NU_SUCCESS`  
The group was successfully added.
- `NU_TIMEOUT`  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- `NU_NO_MEMORY`  
There was not enough memory to satisfy this request.
- `NU_INVALID_PARM`  
Group is being created which already exists.
- `IPSEC_INVALID_PARAMS`  
Input parameter is invalid.

### Example

```
CHAR group_name[] = "MY_GROUP";

/* Add Nucleus IPsec group to the IPsec groups database. */
if (IPSEC_Add_Group(group_name) != NU_SUCCESS)
{
    printf("ERROR: Unable to create an IPsec group.\n");
}
```

### Related Topics

[IPsec API Functions](#)

## IPSEC\_Add\_Inbound\_SA IPSEC\_Add\_Inbound\_ESN\_SA

These functions add a new inbound SA to the Security Associations Database (SADB).

### Usage

```
STATUS IPSEC_Add_Inbound_SA (CHAR          *group_name,  
                             IPSEC_INBOUND_SA *sa_entry,  
                             UINT32          *return_spi);  
  
STATUS IPSEC_Add_Inbound_ESN_SA (CHAR          *group_name,  
                                 IPSEC_INBOUND_SA *sa_entry,  
                                 UINT32          *return_spi);
```

### Arguments

- **group\_name**  
A pointer to the name of the group to which this SA needs to be added.
- **sa\_entry**  
A pointer to the inbound SA to be added. Refer to the [IPSEC\\_INBOUND\\_SA](#) data structure description for further details.
- **return\_spi**  
A pointer to the SPI assigned to the SA by this routine.

### Return Values

- **NU\_SUCCESS**  
The inbound SA was successfully added.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **NU\_NO\_MEMORY**  
There was not enough memory to satisfy this request.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found.
- **NU\_INVALID\_PARM**  
This SA already exists.

## Description

This inbound SA remains in the group unless it is manually removed using the `IPSEC_Remove_Inbound_SA()` routine. It may also be removed indirectly if the associated group is removed. These APIs need not be called if IKE is being used to dynamically establish SAs.

`IPSEC_Add_Inbound_ESN_SA` differs from `IPSEC_Add_Inbound_SA` in that the former creates an SA with Extended Sequence Number (ESN) support.

Memory is allocated for the inbound SA within this function, and the values are copied from the passed structure. Therefore, the passed SA structure can have local scope. This function returns an auto-assigned unique SPI for this SA, if one is not explicitly specified.

### Note



If the caller explicitly specifies an SPI for an inbound SA, then it must lie within the range of 256 to 10000, inclusive. Other SPI values are used internally by Nucleus IPsec.

## Example

```
IPSEC_INBOUND_SA      inbound_sa;
UINT32                in_sa_spi;
CHAR                  group_name[]    = "MY_GROUP";
UINT8                 auth_key[]      = "topsecretkeytest";
UINT8                 encryption_key[] = "sequence";

UINT8  src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8  dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};

/* The following values come as a result of an agreement
   about the keys. */
inbound_sa.ipsec_spi           = 256;
inbound_sa.ipsec_auth_key      = auth_key;
inbound_sa.ipsec_encryption_key = encryption_key;

/* Destination IP address */
memcpy(inbound_sa.ipsec_select.ipsec_source_ip.ipsec_addr,
       dst_ip, IP_ADDR_LEN);

/* Source IP address */
memcpy(inbound_sa.ipsec_select.ipsec_dest_ip.ipsec_addr,
       Src_ip, IP_ADDR_LEN);

/* Destination type should be a single IPv4 address for an SA. */
inbound_sa.ipsec_select.ipsec_source_type =
    (IPSEC_SINGLE_IP | IPSEC_IPV4);

/* Source type should be a single IPv4 address for an SA. */
inbound_sa.ipsec_select.ipsec_dest_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
/* Set the transport protocol, source and destination ports supported by
   this inbound SA. IPSEC_WILDCARD means "any". */
inbound_sa.ipsec_select.ipsec_transport_protocol = IPSEC_WILDCARD;
inbound_sa.ipsec_select.ipsec_source_port        = IPSEC_WILDCARD;
inbound_sa.ipsec_select.ipsec_destination_port   = IPSEC_WILDCARD;
/* Set the inbound SA security parameters. */
```

```
inbound_sa.ipsec_security.ipsec_protocol      = IPSEC_ESP;
inbound_sa.ipsec_security.ipsec_security_mode = IPSEC_TRANSPORT_MODE;
inbound_sa.ipsec_security.ipsec_auth_algo     = IPSEC_HMAC_MD5_96;
inbound_sa.ipsec_security.ipsec_encryption_algo = IPSEC_DES_CBC;
inbound_sa.ipsec_security.ipsec_flags         = IPSEC_IPV4;
/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Add\_Outbound\_SA IPSEC\_Add\_Outbound\_ESN\_SA

These functions add a new outbound SA to the list of outbound SAs.

### Usage

```
STATUS IPSEC_Add_Outbound_SA (CHAR          *group_name,
                             IPSEC_OUTBOUND_SA *sa_entry,
                             UINT32          *return_index);

STATUS IPSEC_Add_Outbound_ESN_SA (CHAR          *group_name,
                                  IPSEC_OUTBOUND_SA *sa_entry,
                                  UINT32          *return_index);
```

### Arguments

- **group\_name**  
A pointer to the name of the group to which this SA is added.
- **sa\_entry**  
A pointer to the outbound SA to be added. Refer to the [IPSEC\\_OUTBOUND\\_SA](#) data structure description for further details.
- **return\_index**  
A pointer to the index assigned to the SA by the function call.

### Return Values

- **NU\_SUCCESS**  
The SA was successfully added.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **NU\_NO\_MEMORY**  
There was not enough memory to satisfy this request.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_INVALID\_ALGO\_ID**  
Algorithm ID provided is incorrect.
- **IPSEC\_NOT\_FOUND**  
The group was not found.

## Description

This outbound SA remains in the group until it is manually removed using the `IPSEC_Remove_Outbound_SA()` routine. It may also be removed indirectly if the associated group is removed. These APIs need not be called if IKE is being used to dynamically establish SAs.

`IPSEC_Add_Outbound_ESN_SA` differs from `IPSEC_Add_Outbound_SA` in that the former creates an SA with Extended Sequence Number (ESN) support.

Memory is allocated for the outbound SA within this function, and the values are copied from the passed structure. Therefore, the passed SA structure can have local scope.

These functions return a locally assigned unique identifier for this SA.

## Example

```
IPSEC_OUTBOUND_SA      outbound_sa;
UINT32                 out_sa_index;
CHAR                   group_name[]    = "MY_GROUP";
UINT8                  auth_key[]      = "topsecretkeytest";
UINT8                  encryption_key[] = "sequence";

UINT8  src_ip[]        = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8  dst_ip[]        = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};

/* Set the SPI, authentication and encryption keys of this SA. */
outbound_sa.ipsec_remote_spi      = 256;
outbound_sa.ipsec_auth_key        = auth_key;
outbound_sa.ipsec_encryption_key  = encryption_key;

/* Destination IP address */
memcpy(outbound_sa.ipsec_select.ipsec_dest_ip.ipsec_addr,
        dst_ip, IP_ADDR_LEN);

/* Source IP address */
memcpy(outbound_sa.ipsec_select.ipsec_source_ip.ipsec_addr,
        src_ip, IP_ADDR_LEN);

/* Destination type should be a single IPv4 address for an SA. */
outbound_sa.ipsec_select.ipsec_dest_type =
    (IPSEC_SINGLE_IP | IPSEC_IPV4);

/* Source type should be a single IPv4 address for an SA. */
outbound_sa.ipsec_select.ipsec_source_type =
    (IPSEC_SINGLE_IP | IPSEC_IPV4);

/* Set the transport protocol, source and destination ports supported
 * by this outbound SA. */
outbound_sa.ipsec_select.ipsec_transport_protocol = IPSEC_WILDCARD;

outbound_sa.ipsec_select.ipsec_destination_port = IPSEC_WILDCARD;
outbound_sa.ipsec_select.ipsec_source_port      = IPSEC_WILDCARD;

/* Set the outbound SA security parameters. */
outbound_sa.ipsec_security.ipsec_protocol      = IPSEC_ESP;
outbound_sa.ipsec_security.ipsec_security_mode = IPSEC_TRANSPORT_MODE;
```



```
outbound_sa.ipsec_security.ipsec_auth_algo      = IPSEC_HMAC_MD5_96;
outbound_sa.ipsec_security.ipsec_encryption_algo = IPSEC_DES_CBC;
outbound_sa.ipsec_security.ipsec_flags          = IPSEC_IPV4;

/* Now create the outbound SA. "out_sa_index" contains the
 * index assigned to this SA.
 */
if(IPSEC_Add_Outbound_SA(group_name, &outbound_sa,
                          &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Add\_Policy

This function adds a new policy to the Security Policy Database (SPDB) of an IPsec group.

### Usage

```
STATUS IPSEC_Add_Policy (CHAR          *group_name,  
                        IPSEC_POLICY *policy,  
                        UINT32        *return_index);
```

### Arguments

- **group\_name**  
A pointer to the name of the group to which this policy is added.
- **policy**  
A pointer to the policy to be added. Refer to the description of the [IPSEC\\_POLICY](#) data structure.
- **return\_index**  
A pointer to the caller's index. On a successful request, the uniquely assigned index is returned.

### Return Values

- **NU\_SUCCESS**  
The policy was successfully added.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **NU\_NO\_MEMORY**  
There was not enough memory to satisfy this request.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found.
- **NU\_INVALID\_PARM**  
Policy already exists.

## Description

Memory is allocated for the policy within this function, and the values will be copied from the passed structure. Therefore, the passed policy structure can have local scope.

This function returns a unique index that is assigned to the policy. The scope of this unique index is within a single group.

When adding multiple IPsec policies, the order in which they are added is significant. For example, if a packet matches two policies in the database, possibly one with a single IP selector and the other with a wildcard selector, then, the policy that was added first is used.

## Example

```
IPSEC_POLICY          new_policy;
IPSEC_SECURITY_PROTOCOL policy_security;
CHAR                  group_name[] = "MY_GROUP";
UINT32                policy_index;

UINT8  src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8  dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};

/* Set the transport protocol, dest and source ports to any. */
new_policy.ipsec_select.ipsec_transport_protocol = IPSEC_WILDCARD;
new_policy.ipsec_select.ipsec_destination_port   = IPSEC_WILDCARD;
new_policy.ipsec_select.ipsec_source_port        = IPSEC_WILDCARD;

/* Destination node type */
new_policy.ipsec_select.ipsec_dest_type =
    (IPSEC_SINGLE_IP | IPSEC_IPV4);

/* Source node type */
new_policy.ipsec_select.ipsec_source_type =
    (IPSEC_SINGLE_IP | IPSEC_IPV4);

/* Copy the destination IP address. */
memcpy(new_policy.ipsec_select.ipsec_dest_ip.ipsec_addr,
        dst_ip, IP_ADDR_LEN);

/* Copy the source IP address. */
memcpy(new_policy.ipsec_select.ipsec_source_ip.ipsec_addr,
        src_ip, IP_ADDR_LEN);

/* Apply security for both inbound and outbound traffic. */
new_policy.ipsec_flags = (IPSEC_APPLY | IPSEC_DUAL_ASYNCHRONOUS);

/* Set the security parameters. */
policy_security.ipsec_protocol      = IPSEC_ESP;
policy_security.ipsec_auth_algo     = IPSEC_HMAC_MD5_96;
policy_security.ipsec_encryption_algo = IPSEC_DES_CBC;
policy_security.ipsec_flags         = IPSEC_IPV4;
policy_security.ipsec_sa_derivation = IPSEC_VALUE_FROM_POLICY;
policy_security.ipsec_security_mode = IPSEC_TRANSPORT_MODE;

/* Now add this security to the policy security array. */
new_policy.ipsec_security = &policy_security;
```

```
/* Only one security protocol is going to be applied. */
new_policy.ipsec_security_size = 1;

/* The following three values are set for IKE. */
new_policy.ipsec_sa_max_lifetime.ipsec_expiry_action =
    IPSEC_REFRESH_SA;

new_policy.ipsec_sa_max_lifetime.ipsec_no_of_secs = 100;
new_policy.ipsec_pfs_group_desc = IKE_GROUP_NONE;

/* Set IPsec SA bundle lifetime in seconds. All the SA bundles
 * within the policy share this value.
 */
new_policy.ipsec_bundles.ipsec_lifetime = 172800;

/* Add this policy to the group. policy_index contains the
 * index assigned to this policy.
 */
if(IPSEC_Add_Policy(group_name, &new_policy,
    &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the policy to the group.\r\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Add\_To\_Group

This function registers an interface with a Nucleus IPsec group.

### Usage

```
STATUS IPSEC_Add_To_Group (CHAR *group_name,  
                           CHAR *interface_name);
```

### Arguments

- **group\_name**  
A pointer to the name of the group to which the interface is to be added.
- **interface\_name**  
A pointer to the name of a valid interface that is to be registered with the provided group. If the interface is already registered with some existing group, an error is returned.

### Return Values

- **NU\_SUCCESS**  
The interface has been successfully associated with the respective group.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid. A possible reason could be that the interface is already assigned to another group.
- **IPSEC\_NOT\_FOUND**  
The group or the interface was not found.
- **IPSEC\_ALREADY\_EXIST**  
Interface is already registered with other group.

### Description

All the IPsec processing required on the traffic passing through this interface is based on this group and its sub-members.

Any interface that requires IPsec processing on its traffic must first be registered with some valid Nucleus IPsec group using this API. An interface may be registered with only one IPsec group.

### Example

```
NU_DEVICE devices[1];  
CHAR group_name[] = "MY_GROUP";  
devices[0].dv_name = "if_name";  
.
```

```
.  
.   
  
/* Initialize the interface. */  
status = NU_Init_Devices(devices, 1);  
  
if (status != NU_SUCCESS)  
{  
    printf("ERROR: Unable to initialize the interface.\n");  
}  
  
/* Add Nucleus IPsec group to the IPsec group database. */  
if(IPSEC_Add_Group(group_name) != NU_SUCCESS)  
{  
    printf("ERROR: Unable to create an IPsec group.\n");  
}  
  
/* Register the initialized interface with the IPsec group. */  
if(IPSEC_Add_To_Group(group_name, "if_name") != NU_SUCCESS)  
{  
    printf("ERROR: Unable to add the device to the group.\n");  
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Apply\_To\_Interface

This function enables or disables IPsec on an interface.

### Usage

```
STATUS IPSEC_Apply_To_Interface (CHAR  *if_name,  
                                UINT8  state_flag);
```

### Arguments

- **if\_name**  
A pointer to the name of the interface on which IPsec processing is being enabled or disabled.
- **state\_flag**  
A flag indicating whether to enable or disable IPsec on the interface. Valid values are IPSEC\_ENABLE and IPSEC\_DISABLE.

### Return Values

- **NU\_SUCCESS**  
The request has been successfully executed.
- **IPSEC\_NOT\_FOUND**  
An interface with the given name is not present, or the interface has not been registered with an IPsec group.
- **IPSEC\_INVALID\_PARAMS**  
Some invalid parameter has been passed.

### Description

If IPsec is disabled on an interface, then no security checks are applied on incoming and outgoing traffic passing through the interface, and it behaves as a usual (non-IPsec) interface. However, if enabled, every packet is processed with IPsec checks.

---

#### Note



Nucleus NET, Nucleus IPsec, and the respective interface must be initialized before calling this routine. Also, the interface must first be registered with an IPsec group via the API routine IPSEC\_Add\_To\_Group() before enabling IPsec security services on the interface.

---

### Example - Enabling IPsec

```
CHAR  group_name[] = "MY_GROUP";  
/* Initialize Nucleus NET and Nucleus IPsec. */  
. . .  
  
/* Add a Nucleus IPsec group to the Nucleus IPsec group database.*/  
if(IPSEC_Add_Group(group_name) != NU_SUCCESS)
```

```
{
    printf("ERROR: Unable to create an IPsec group.\n");
}

/* Now register the desired interface with the above group. */
if(IPSEC_Add_To_Group(group_name, "if_name") != NU_SUCCESS)
{
    printf("ERROR: Unable to register the interface with the group.\n");
}

/* Now enable IPsec on the interface. */
if (IPSEC_Apply_To_Interface("if_name", IPSEC_ENABLE) != NU_SUCCESS)
{
    printf("ERROR: Unable to enable IPsec on the interface.\n");
}
```

### Example – Disabling IPsec

```
/* Disable IPsec on the interface. */
if (IPSEC_Apply_To_Interface("if_name", IPSEC_DISABLE) != NU_SUCCESS)
{
    printf("ERROR: Unable to disable IPsec processing
           on the interface.\n");
}
```

### Related Topics

[IPsec API Functions](#)



## IPSEC\_Get\_Group

This function returns the group name for the given interface. If the interface is not registered with any IPsec group, the function returns an error.

### Usage

```
STATUS IPSEC_Get_Group (CHAR    *interface_name,  
                        CHAR    *return_group,  
                        UINT32  *total_len);
```

### Arguments

- **interface\_name**  
A pointer to the name of the interface.
- **return\_group**  
On successful completion, this points to the name of the group with which the interface is registered.
- **total\_len**  
The maximum size string that can be placed in return\_group. On successful completion, this contains the size of the name copied into return\_group. The given length must include the terminating null character and must be large enough to store the group name.

### Return Values

- **NU\_SUCCESS**  
The operation was successfully completed, and return\_group contains the name of the group with which the interface is associated.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_NOT\_FOUND**  
The interface was not found.
- **IPSEC\_INVALID\_PARAMETERS**  
One of the input parameters is invalid. A possible reason could be that the interface name is not valid.
- **IPSEC\_LENGTH\_IS\_SHORT**  
Given length is short.

### Example

```
#define MAX_GROUP_NAME_LEN 20  
  
CHAR    group_name[MAX_GROUP_NAME_LEN];  
UINT32  group_name_len = MAX_GROUP_NAME_LEN;
```

```
/* Get the IPsec group name with which the passed interface is registered.
*/
if(IPSEC_Get_Group("if_name", group_name,
                  &group_name_len) != NU_SUCCESS)
{
    printf("ERROR: Unable get the group name.\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Get\_Group\_Opt

This function returns the IPsec group values as specified by optname.

### Usage

```
STATUS IPSEC_Get_Group_Opt (CHAR  *group_name,
                           INT    optname,
                           VOID   *optval,
                           INT    *optlen);
```

### Arguments

- group\_name

A pointer to the name of the group for which to return the specified values.

- optname

The option to return. Valid values are:

#### IPSEC\_DF\_PROCESSING

A flag that specifies the group's treatment of the DF bit (set, clear, copy from encapsulated header). A UINT8 value is returned in the optval parameter. Valid values are:

##### IPSEC\_SET\_DF\_BIT

The DF bit in the tunneling IP header is always set.

##### IPSEC\_CLEAR\_DF\_BIT

The DF bit in the tunneling IP header is never set.

##### IPSEC\_COPY\_DF\_BIT

The DF bit in the tunneling IP header is copied from the tunneled header.

#### IPSEC\_IS\_GROUP

Check whether the passed name identifies a group. The routine returns NU\_SUCCESS if the group is a valid group. optval and optlen are NU\_NULL.

#### IPSEC\_NEXT\_GROUP

Returns the name of the next group in the list following this group. The groups are sorted in ascending order. A null-terminated CHAR\* value is returned in the optval parameter.

- optval

A pointer to the location where the value is placed on a successful request.

- optlen

The total length, in bytes, of the memory pointed to by optval. On return, the number of bytes that have been written to optval. If the given length is less than the required option value length, the required length is returned in this parameter.

## Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMETERS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found.
- **IPSEC\_LENGTH\_IS\_SHORT**  
The value to be returned requires more memory than was passed. The number of bytes that was required to be written is returned in optlen.

## Example

```
#define MAX_GROUP_LEN 100

CHAR group_name[] = "MY_GROUP";
UINT8 df_setting;
INT df_setting_size = sizeof(UINT8);
UINT8 next_group_name[MAX_GROUP_LEN];
INT next_group_name_len = MAX_GROUP_LEN;

/* Get the group treatment of the DF bit option. */
if(IPSEC_Get_Group_Opt(group_name, IPSEC_DF_PROCESSING,
    (VOID*)&df_setting,
    &df_setting_size) != NU_SUCCESS)
{
    printf("ERROR: Unable to get the DF bit option.\n");
}

/* Check the validity of the group name. */
if(IPSEC_Get_Group_Opt(group_name, IPSEC_IS_GROUP, OS_NULL,
    OS_NULL) != NU_SUCCESS)
{
    printf("ERROR: Passed group is not a valid IPsec group.\n");
}

/* Get next group name. */
if(IPSEC_Get_Group_Opt(group_name, IPSEC_NEXT_GROUP,
    (VOID*)next_group_name,
    &next_group_name_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get the next group name.\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Get\_Inbound\_SA\_Opt

This function returns the characteristics of an inbound SA value as specified by optname.

### Usage

```
STATUS IPSEC_Get_Inbound_SA_Opt (CHAR          *group_name,  
                                IPSEC_INBOUND_INDEX *index,  
                                INT              optname,  
                                VOID             *optval,  
                                INT              optlen);
```

### Arguments

- **group\_name**  
A pointer to the IPsec group with which the intended SA is associated.
- **index**  
A pointer to an identifier that uniquely identifies the inbound SA. Refer to the [IPSEC\\_INBOUND\\_INDEX](#) data structure description for more information.

- **optname**

The option to retrieve. Valid values are:

#### IPSEC\_SECURITY

Returns the security protocol data structure associated with the SA. The optval parameter contains an [Related Topics](#) data structure.

#### IPSEC\_SELECT

Returns the selector associated with this SA. The optval parameter contains an [IPSEC\\_SELECTOR](#) data structure.

#### IPSEC\_SOFT\_LIFETIME

Returns the soft lifetime associated with the SA. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. Only valid if IKE is included in the build.

#### IPSEC\_HARD\_LIFETIME

Returns the hard lifetime associated with the SA. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. Only valid if IKE is included in the build.

#### IPSEC\_AUTH\_KEY

Returns the authentication key associated with this SA. The optval parameter contains a CHAR pointer into memory.

#### IPSEC\_ENCRYPTION\_KEY

Returns the encryption key associated with this SA. The optval parameter contains a CHAR pointer into memory.

#### IPSEC\_IS\_SA

Check whether the passed index is an SA. Upon success, this routine returns NU\_SUCCESS, and the optval parameter is NU\_NULL.

## IPSEC\_NEXT\_SA

Returns the index for the next SA after the SA specified by the passed index. If there is no next SA, the index for the first SA is returned. The optval parameter contains a UINT32 value.

- **optval**  
A pointer to the location where the value is placed on a successful request.
- **optlen**  
The total length, in bytes, of the memory pointed to by optval. On return, contains the number of bytes that have been written to optval.

## Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMETERS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found, or there is no SA corresponding to the passed index.
- **IPSEC\_LENGTH\_IS\_SHORT**  
The value to be returned requires more memory than was provided. The number of bytes required to be written is returned in optlen.

## Example – IPSEC\_SECURITY Option

```
IPSEC_INBOUND_INDEX      sa_index;
IPSEC_SECURITY_PROTOCOL  sa_security;
INT                       sa_security_len;
UINT8                    dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA         inbound_sa;
UINT32                   in_sa_spi;
CHAR                     group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
```

```
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;
/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));
/* Get the security structure size. */
sa_security_len = sizeof(IPSEC_SECURITY_PROTOCOL);

/* Get the inbound SA security protocol. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_SECURITY,
                           (VOID*)&sa_security,
                           &sa_security_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SECURITY option.\n");
}
```

### Example – IPSEC\_SELECT Option

```
IPSEC_INBOUND_INDEX    sa_index;
IPSEC_SELECTOR         sa_selector;
INT                   sa_selector_len;
UINT8                 dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};

IPSEC_INBOUND_SA      inbound_sa;
UINT32                in_sa_spi;
CHAR                  group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

/* Set the inbound SA index. */
sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the selector structure size. */
sa_selector_len = sizeof(IPSEC_SELECTOR);

/* Get the inbound SA selector. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_SELECT,
                           (VOID*)&sa_selector,
```



```

                                &sa_selector_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SELECT option.\n");
}

```

### Example – IPSEC\_SOFT\_LIFETIME Option

```

IPSEC_INBOUND_INDEX      sa_index;
IPSEC_SA_LIFETIME        sa_soft_lifetime;
INT                       sa_soft_lifetime_len;
UINT8 dst_ip[]           = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA         inbound_sa;
UINT32                   in_sa_spi;
CHAR                     group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}
sa_index.ipsec_spi        = in_sa_spi;
sa_index.ipsec_dest_type  = IPSEC_IPV4;
sa_index.ipsec_protocol   = IPSEC_AH;
sa_soft_lifetime_len      = sizeof(IPSEC_SA_LIFETIME);

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the inbound SA option. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_SOFT_LIFETIME,
                            (VOID*)&sa_soft_lifetime,
                            &sa_soft_lifetime_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SOFT_LIFETIME option.\n");
}

```

### Example – IPSEC\_HARD\_LIFETIME Option

```

IPSEC_INBOUND_INDEX      sa_index;
IPSEC_SA_LIFETIME        sa_hard_lifetime;
INT                       sa_hard_lifetime_len;
UINT8 dst_ip[]           = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA         inbound_sa;
UINT32                   in_sa_spi;
CHAR                     group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned

```

```

    to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}
sa_index.ipsec_spi= in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;
sa_hard_lifetime_len    = sizeof(IPSEC_SA_LIFETIME);

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the hard lifetime SA option. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index,
                            IPSEC_HARD_LIFETIME,
                            (VOID*)&sa_hard_lifetime,
                            &sa_hard_lifetime_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_HARD_LIFETIME option.\n");
}

```

### Example – IPSEC\_AUTH\_KEY Option

```

#define BUFF_SIZE      100
IPSEC_INBOUND_INDEX    sa_index;
UINT8                  auth_key[BUFF_SIZE];
INT                    auth_key_buff_len;
UINT8 dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA       inbound_sa;
UINT32                 in_sa_spi;
CHAR                   group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;
auth_key_buff_len       = BUFF_SIZE;

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the SA's authentication key. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_AUTH_KEY,
                            (VOID*)auth_key,

```

```

                                &auth_key_buff_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_AUTH_KEY option.\n");
}

```

### Example - IPSEC\_ENCRYPTION\_KEY Option

```

#define BUFF_SIZE          100
IPSEC_INBOUND_INDEX       sa_index;
UINT8                     encryp_key[BUFF_SIZE];
INT                       encryp_key_buff_len;
UINT8 dst_ip[]            = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA          inbound_sa;
UINT32                    in_sa_spi;
CHAR                      group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi        = in_sa_spi;
sa_index.ipsec_protocol   = IPSEC_AH;
sa_index.ipsec_dest_type  = IPSEC_IPV4;
encryp_key_buff_len       = BUFF_SIZE;

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the SA's encryption key length. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_ENCRYPTION_KEY,
                            (VOID*)encryp_key,
                            &encryp_key_buff_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_ENCRYPTION_KEY option.\n");
}

```

### Example – IPSEC\_IS\_SA Option

```

IPSEC_INBOUND_INDEX       sa_index;
UINT8 dst_ip[]            = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA          inbound_sa;
UINT32                    in_sa_spi;
CHAR                      group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

```

```

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Validate the passed SA index. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_IS_SA,
                           OS_NULL, OS_NULL) != NU_SUCCESS)
{
    printf("ERROR: sa_index is not a valid inbound SA\n");
}

```

### Example – IPSEC\_NEXT\_SA Option

```

IPSEC_INBOUND_INDEX    sa_index;
IPSEC_INBOUND_INDEX    next_sa;
INT                    next_sa_len;
UINT8 dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
IPSEC_INBOUND_SA       inbound_sa;
UINT32                 in_sa_spi;
CHAR                   group_name[] = "MY_GROUP";

/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;
next_sa_len             = sizeof(IPSEC_INBOUND_INDEX);

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Get the next SA index to the passed index. */
if(IPSEC_Get_Inbound_SA_Opt("MY_GROUP", &sa_index, IPSEC_NEXT_SA,
                           (VOID*)&next_sa,
                           &next_sa_len) != NU_SUCCESS)

```

```
{  
    printf("ERROR: Unable to get IPSEC_NEXT_SA option.\n");  
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Get\_Outbound\_SA\_Opt

This function returns information associated with an outbound SA as specified by optname.

### Usage

```
STATUS IPSEC_Get_Outbound_SA_Opt (IPSEC_OUTBOUND_INDEX *index,  
                                  INT optname,  
                                  VOID *optval,  
                                  INT *optlen)
```

### Arguments

- index

A pointer to an identifier that uniquely identifies the outbound SA. Refer to the description of the [IPSEC\\_OUTBOUND\\_INDEX](#) for more information on this data structure.

- optname

The option to retrieve. Valid values are:

#### IPSEC\_SECURITY

Returns the SA security protocol structure. The optval parameter contains an [Related Topics](#) data structure.

#### IPSEC\_SELECT

Returns the SA selector. The optval parameter contains an [IPSEC\\_SELECTOR](#) data structure.

#### IPSEC\_SOFT\_LIFETIME

Returns the soft lifetime structure associated with this SA. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. This option is supported only if IKE is included in the build.

#### IPSEC\_HARD\_LIFETIME

Returns the hard lifetime structure associated with this SA. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. This option is supported only if IKE is included in the build.

#### IPSEC\_AUTH\_KEY

Returns the authentication key associated with this SA. The optval parameter contains a UINT8 pointer to memory.

#### IPSEC\_ENCRYPTION\_KEY

Returns the encryption key associated with this SA. The optval parameter contains a UINT8 pointer to memory.

#### IPSEC\_IS\_SA

Checks whether the passed index is for a valid SA. This routine returns NU\_SUCCESS if the SA valid, and optval is NU\_NULL.

#### IPSEC\_NEXT\_SA

Returns the index for the SA located after the SA provided. If there is no next SA, then the index for the first SA is returned. The optval parameter contains the SA index of UINT32 data type.

- optval  
A pointer to the location into which the value is placed.
- optlen  
The total length, in bytes, of the memory pointed to by optval. On return, contains the number of bytes that have been written to optval.

### Return Values

- NU\_SUCCESS  
The request was successfully executed.
- NU\_TIMEOUT  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- IPSEC\_INVALID\_PARAMS  
One of the input parameters is invalid.
- IPSEC\_NOT\_FOUND  
The group was not found, or there is no SA corresponding to the passed index.
- IPSEC\_LENGTH\_IS\_SHORT  
The value to be returned requires more memory than was provided. The number of bytes that had to be written are returned in optlen.

### Example – IPSEC\_SECURITY Option

```
IPSEC_OUTBOUND_INDEX    sa_index;
IPSEC_SECURITY_PROTOCOL sa_security;
INT                     sa_security_len;
IPSEC_OUTBOUND_SA      outbound_sa;
UINT32                 out_sa_index;
CHAR                    group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any IPsec group.
*/
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}
```

```

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index= out_sa_index;
sa_security_len = sizeof(IPSEC_SECURITY_PROTOCOL);

/* Get the desired outbound SA option. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_SECURITY,
                             (VOID*)&sa_security,
                             &sa_security_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SECURITY option.\n");
}

```

### Example – IPSEC\_SELECT Option

```

IPSEC_OUTBOUND_INDEX      sa_index;
IPSEC_SELECTOR            sa_selector;
INT                       sa_selector_len;
IPSEC_OUTBOUND_SA        outbound_sa;
UINT32                   out_sa_index;
CHAR                      group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any
   IPsec group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                         &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index= out_sa_index;
sa_selector_len = sizeof(IPSEC_SELECTOR);

/* Get the required option. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_SELECT,
                             (VOID*)&sa_selector,
                             &sa_selector_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SELECT option.\n");
}

```

### Example – IPSEC\_SOFT\_LIFETIME Option

```

IPSEC_OUTBOUND_INDEX      sa_index;
IPSEC_SA_LIFETIME         sa_soft_lifetime;
INT                       sa_soft_lifetime_len;
IPSEC_OUTBOUND_SA        outbound_sa;
UINT32                   out_sa_index;
CHAR                      group_name[] = "MY_GROUP";

```



```

/* Set all the fields of outbound SA before adding it to any
   IPsec group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index= out_sa_index;;
sa_soft_lifetime_len = sizeof(IPSEC_SA_LIFETIME);

/* Get the soft lifetime value. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_SOFT_LIFETIME,
                            (VOID*)&sa_soft_lifetime,
                            &sa_soft_lifetime_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SOFT_LIFETIME option.\n");
}

```

### Example – IPSEC\_HARD\_LIFETIME Option

```

IPSEC_OUTBOUND_INDEX    sa_index;
IPSEC_SA_LIFETIME       sa_hard_lifetime;
INT                     sa_hard_lifetime_len;
IPSEC_OUTBOUND_SA       outbound_sa;
UINT32                  out_sa_index;
CHAR                    group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any
   IPsec group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index= out_sa_index;
sa_hard_lifetime_len = sizeof(IPSEC_SA_LIFETIME);

/* Get the hard lifetime option. */

```

```
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_HARD_LIFETIME,
                             (VOID*)&sa_hard_lifetime,
                             &sa_hard_lifetime_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_HARD_LIFETIME option.\n");
}
```

### Example – IPSEC\_AUTH\_KEY Option

```
#define BUFF_SIZE      100
IPSEC_OUTBOUND_INDEX  sa_index;
UINT8                 auth_key[BUFF_SIZE];
INT                   auth_key_buff_len;
IPSEC_OUTBOUND_SA     outbound_sa;
UINT32                out_sa_index;
CHAR                  group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any
   IPsec group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index = out_sa_index;
auth_key_buff_len    = BUFF_SIZE;

/* Get the authentication key. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_AUTH_KEY,
                             (VOID*)auth_key,
                             &auth_key_buff_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_AUTH_KEY option.\n");
}
```

### Example - IPSEC\_ENCRYPTION\_KEY Option

```
#define BUFF_SIZE      100

IPSEC_OUTBOUND_INDEX  sa_index;
UINT8                 encryp_key[BUFF_SIZE];
INT                   encryp_key_buff_len;
IPSEC_OUTBOUND_SA     outbound_sa;
UINT32                out_sa_index;
CHAR                  group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any IPsec
   group. */
.
```

```
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index = out_sa_index;
encryp_key_buff_len  = BUFF_SIZE;

/* Get the encryption key. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_ENCRYPTION_KEY,
                            (VOID*)encryp_key,
                            &encryp_key_buff_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_ENCRYPTION_KEY option.\n");
}
```

### Example – IPSEC\_IS\_SA Option

```
IPSEC_OUTBOUND_INDEX    sa_index;
IPSEC_OUTBOUND_SA       outbound_sa;
UINT32                  out_sa_index;
CHAR                    group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any
   IPsec
   group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index= out_sa_index;

/* Check whether the passed index corresponds to some valid SA. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_IS_SA,
                            OS_NULL, OS_NULL) != NU_SUCCESS)
{
    printf("ERROR: sa_index is not a valid outbound SA\n");
}
```

## Example – IPSEC\_NEXT\_SA Option

```
IPSEC_OUTBOUND_INDEX    sa_index;
UINT32                  next_sa_index;
INT                      next_sa_len;
IPSEC_OUTBOUND_SA       outbound_sa;
UINT32                  out_sa_index;
CHAR                    group_name[] = "MY_GROUP";

/* Set all the fields of outbound SA before adding it to any IPsec
   group. */
.
.
.

/* Now create the outbound SA. "out_sa_index" contains the
   index assigned to this SA. */
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,
                        &out_sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the outbound SA. \n");
}

/* Set the index parameters. */
sa_index.ipsec_group = group_name;
sa_index.ipsec_index = out_sa_index;
next_sa_len          = sizeof(UINT32);

/* Get the required SA option value. */
if(IPSEC_Get_Outbound_SA_Opt(&sa_index, IPSEC_NEXT_SA,
                            (VOID*)&next_sa_index,
                            &next_sa_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_NEXT_SA option.\n");
}
```

## Related Topics

### [IPsec API Functions](#)

## IPSEC\_Get\_Policy\_Index

This function returns a policy index matching the passed selector in the passed group. If the selector matches multiple policies in the database, the first matched policy is returned.

### Usage

```
STATUS IPSEC_Get_Policy_Index (CHAR          *group_name,  
                              IPSEC_SELECTOR *selector,  
                              UINT8         selector_type,  
                              UINT32        *return_index)
```

### Arguments

- **group\_name**  
A pointer to the name of the group for which the policy is to be found.
- **selector**  
A pointer to the selection criteria for which the search is to be made.
- **selector\_type**  
Indicates whether the selector is inbound or outbound. Valid values are `IPSEC_INBOUND` or `IPSEC_OUTBOUND`. The [IPSEC\\_SELECTOR](#) data structure is described in the [IPsec Data Structures](#) section.
- **return\_index**  
A pointer to the location where the index is placed on a successful request.

### Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found, or there is no policy that matches the passed selection criteria.

### Example

```
UINT8  src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};  
UINT8  dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};  
  
CHAR          group_name[] = "MY_GROUP";  
UINT32        policy_index;  
IPSEC_SELECTOR policy_selector;
```

```
/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;

policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
policy_selector.ipsec_source_port        = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip,
        IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector, IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Get\_Policy\_Opt

This function returns the requested values of a group as specified by optname.

### Usage

```
STATUS IPSEC_Get_Policy_Opt (CHAR    *group_name,  
                             UINT32  index,  
                             INT      optname,  
                             VOID     *optval,  
                             INT      *optlen);
```

### Arguments

- group\_name

A pointer to the name of the group to which the policy is associated.

- index

The uniquely assigned policy index.

- optname

The option to retrieve. Valid values are:

#### IPSEC\_SECURITY

Returns the policy security array of data type [IPSEC\\_SECURITY\\_PROTOCOL](#) in optval. The size of this security array could be zero or more than zero.

#### IPSEC\_SELECT

Returns the policy selector. The optval parameter contains an [IPSEC\\_SELECTOR](#) data structure.

#### IPSEC\_FLAGS

Returns the policy flag values indicating the action and direction of the policy. The optval parameter contains a UINT8 value.

#### IPSEC\_IS\_POLICY

Check whether the passed index is a policy, this returns NU\_SUCCESS, and the optval parameter is NU\_NULL.

#### IPSEC\_NEXT\_POLICY

Returns the next policy index if any exist. Otherwise, the index of the first policy in the group is returned. The optval parameter contains a UINT32 value.

#### IPSEC\_LIFETIME

Returns the lifetime of all the SAs that are established based on this policy. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. Valid only if IKE is included in the build.

#### IPSEC\_SA\_BUNDLE\_LIFETIME

Returns the lifetime associated with the bundle list. The optval parameter contains a UINT32 value.

### IPSEC\_PFS\_GROUP\_DESC

Returns the Perfect Forward Secrecy (PFS) group associated with the policy. The optval parameter contains a UINT16 value. Valid only if IKE is included in the build.

- **optval**  
A pointer to the location where the value is placed on a successful request.
- **optlen**  
Contains the total length, in bytes, of memory pointed to by optval. On return, contains the number of bytes that have been written to optval, or, if the required value length is greater than size of the passed value location, required length is returned.

### Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found, or there is no policy corresponding to the passed index.
- **IPSEC\_LENGTH\_IS\_SHORT**  
The value to be returned requires more memory than was passed. The number of bytes that were required to be written is returned in optlen.

### Example – IPSEC\_SECURITY Option

```
UINT8  src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8  dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};

CHAR                                         group_name[] = "MY_GROUP";
UINT32                                     policy_index;
IPSEC_SELECTOR                             policy_selector;
IPSEC_SECURITY_PROTOCOL policy_security;
INT                                         policy_security_len;

/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);
policy_selector.selector_type     = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port  = IPSEC_WILDCARD;
policy_selector.ipsec_source_port       = IPSEC_WILDCARD;
```



```

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip,
        IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip,
        IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                        IPSEC_OUTBOUND,
                        &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Get the size of the security protocol. */
policy_security_len = sizeof(IPSEC_SECURITY_PROTOCOL);

/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index, IPSEC_SECURITY,
                        (VOID*)&policy_security,
                        &policy_security_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_SECURITY option.\n");
}

```

### Example – IPSEC\_SELECT Option

```

CHAR            group_name[] = "MY_GROUP";
IPSEC_SELECTOR  policy_selector;
INT             policy_selector_len;
UINT8          src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8          dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32         policy_index;
/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type    = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
policy_selector.ipsec_source_port        = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                        IPSEC_OUTBOUND,
                        &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}
policy_selector_len = sizeof(IPSEC_SELECTOR);

/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index, IPSEC_SELECT,
                        (VOID*)&policy_selector,
                        &policy_selector_len) != NU_SUCCESS)

```

```
{
    printf("ERROR: Unable to get IPSEC_SELECT option.\n");
}
```

### Example – IPSEC\_FLAGS Option

```
CHAR            group_name[] = "MY_GROUP";
UINT8           policy_flags;
INT             policy_flags_len;
IPSEC_SELECTOR  policy_selector;
UINT8           src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8           dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32          policy_index;
/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
policy_selector.ipsec_source_port        = IPSEC_WILDCARD;
memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Get the length of "policy_flags" member. */
policy_flags_len = sizeof(UINT8);

/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index, IPSEC_FLAGS,
                        (VOID*)&policy_flags,
                        &policy_flags_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_FLAGS option.\n");
}
```

### Example – IPSEC\_IS\_POLICY Option

```
CHAR            group_name[] = "MY_GROUP";
UINT8           src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8           dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32          policy_index;
IPSEC_SELECTOR  policy_selector;

/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
```

```
policy_selector.ipsec_source_port      = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index, IPSEC_IS_POLICY,
                       OS_NULL, OS_NULL) != NU_SUCCESS)
{
    printf("ERROR: poilcy index is not a valid policy.\n");
}
```

### Example – IPSEC\_LIFETIME Option

```
CHAR          group_name[] = "MY_GROUP";
IPSEC_SA_LIFETIME policy_sa_lifetime;
INT           policy_sa_lifetime_len;
UINT8         src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8         dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32        policy_index;
IPSEC_SELECTOR policy_selector;

/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type     = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
policy_selector.ipsec_source_port        = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);
/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Get the length of the "IPSEC_SA_LIFETIME" structure. */
policy_sa_lifetime_len = sizeof(IPSEC_SA_LIFETIME);
/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index,
                       IPSEC_LIFETIME, (VOID*)&policy_sa_lifetime,
                       &policy_sa_lifetime_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_LIFETIME option.\n");
}
```

## Example – IPSEC\_PFS\_GROUP\_DESC Option

```
CHAR          group_name[] = "MY_GROUP";
UINT16        policy_pfs_group_desc;
INT           policy_pfs_group_desc_len;
UINT8         src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8         dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32        policy_index;
IPSEC_SELECTOR policy_selector;

/* Initializing policy selector to look up policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type     = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port   = IPSEC_WILDCARD;
policy_selector.ipsec_source_port        = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);
/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}
policy_pfs_group_desc_len = sizeof(UINT16);

/* Get the desired policy option. */
if(IPSEC_Get_Policy_Opt(group_name, policy_index,
                        IPSEC_PFS_GROUP_DESC,
                        (VOID*)&policy_pfs_group_desc,
                        &policy_pfs_group_desc_len) != NU_SUCCESS)
{
    printf("ERROR: Unable to get IPSEC_PFS_GROUP_DESC option.\n");
}
```

## Related Topics

### [IPsec API Functions](#)

## IPSEC\_Remove\_From\_Group

This function disassociates an interface from the IPsec group with which it was last registered. It returns an error if the interface is not registered with an IPsec group.

As a result of this call, all incoming and outgoing traffic through this interface is dropped, since the interface is no longer registered with an IPsec group.

### Usage

```
STATUS IPSEC_Remove_From_Group (CHAR *interface_name);
```

### Arguments

- **interface\_name**  
A pointer to the name of the interface to be removed from the IPsec group with which it is registered.

### Return Values

- **NU\_SUCCESS**  
The interface was successfully disassociated with the IPsec group with which it was registered.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_NOT\_FOUND**  
The given interface was not found, or the interface is not registered with an IPsec group.
- **IPSEC\_INVALID\_PARAMETERS**  
The input parameter is invalid. A possible reason could be that the interface name is not valid.

### Example

```
/* Remove the interface from the group with which it is
   registered. */
if(IPSEC_Remove_From_Group("if_name") != NU_SUCCESS)
{
    printf("ERROR: Unable to disassociate the interface from the IPsec
           group with which it is registered.\n");
}
```

### Related Topics

[IPsec API Functions](#)

## IPSEC\_Remove\_Group

This function removes an IPsec group from the database of IPsec groups. All interfaces registered with the given group are disassociated from the group, causing their inbound and outbound traffic to be immediately dropped. The Security Policy Database (SPDB) and inbound and outbound Security Association Databases (SADB) attached to this group are flushed before this group is completely removed from the IPsec group database.

### Usage

```
STATUS IPSEC_Remove_Group (CHAR *group_name);
```

### Arguments

- `group_name`  
A pointer to the name of the group to be removed.

### Return Values

- `NU_SUCCESS`  
The group was successfully deleted.
- `NU_TIMEOUT`  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- `IPSEC_NOT_FOUND`  
The group was not found.
- `IPSEC_INVALID_PARAMS`  
The input parameter is invalid.

### Example

```
CHAR group_name[] = "MY_GROUP";

/* Remove the group from IPsec groups database. */
if (IPSEC_Remove_Group(group_name) != NU_SUCCESS)
{
    printf("ERROR: Unable to remove the group.\n");
}
```

### Related Topics

[IPsec API Functions](#)

## IPSEC\_Remove\_Inbound\_SA

This function removes an inbound SA as specified by the index.

### Usage

```
STATUS IPSEC_Remove_Inbound_SA (CHAR *group_name,
                                IPSEC_INBOUND_INDEX *index);
```

### Arguments

- **group\_name**  
A pointer to the IPsec group to which this SA belongs.
- **index**  
A pointer to an identifier that uniquely identifies the inbound SA to be removed. Refer to the description of [IPSEC\\_INBOUND\\_INDEX](#) for more information.

### Return Values

- **NU\_SUCCESS**  
The inbound SA was successfully removed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_NOT\_FOUND**  
The group was not found or there is no SA corresponding to the passed index.
- **IPSEC\_INVALID\_PARAMETERS**  
One of the input parameters is invalid.

### Description

Only SAs which have been manually added can be removed using this routine. IKE negotiated SAs cannot be removed through this routine, but are only removed by the system when their lifetime is finished. The lifetime is assigned to an IKE negotiated SA after mutual agreement between the two nodes establishing the SA for one-to-one communication. Refer to the [IPsec-IKE](#) chapter for further information.

The SA is immediately removed from the group even if it is being used actively, and all the traffic based on the SA is dropped unless another SA with the same credentials already exists, or is added to the same group manually or is negotiated through IKE.

### Example

```
IPSEC_INBOUND_INDEX    sa_index;
UINT8 dst_ip[]         = { (CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20 };
IPSEC_INBOUND_SA       inbound_sa;
UINT32                 in_sa_spi;
CHAR                    group_name[] = "MY_GROUP";
```

```
/* Set all the required fields of inbound SA to be added. */
.
.
.

/* Add the inbound SA. "in_sa_spi" contains the index assigned
   to this SA. */
if(IPSEC_Add_Inbound_SA(group_name, &inbound_sa,
                        &in_sa_spi) != NU_SUCCESS)
{
    printf("ERROR: Unable to add the Inbound SA.\n");
}

sa_index.ipsec_spi      = in_sa_spi;
sa_index.ipsec_protocol = IPSEC_AH;
sa_index.ipsec_dest_type = IPSEC_IPV4;

/* Set the destination address. */
memcpy(sa_index.ipsec_dest, dst_ip, sizeof(dst_ip));

/* Remove the SA with the given index. */
if(IPSEC_Remove_Inbound_SA("MY_GROUP", &sa_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to remove the SA.\n");
}
```

## Related Topics

### [IPsec API Functions](#)



## IPSEC\_Remove\_Outbound\_SA

This function removes the specified outbound SA.

### Usage

```
STATUS IPSEC_Remove_Outbound_SA (IPSEC_OUTBOUND_INDEX *index);
```

### Arguments

- **index**  
A pointer to an identifier that uniquely identifies an outbound SA that is to be removed. Refer to the description of the [IPSEC\\_OUTBOUND\\_INDEX](#) data structure.

### Return Values

- **NU\_SUCCESS**  
The outbound SA was successfully removed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_NOT\_FOUND**  
The group was not found, or there is no SA corresponding to the passed index.
- **IPSEC\_INVALID\_PARAMS**  
One of the parameter(s) is invalid.

### Description

Only SAs which have been added manually can be removed using this routine. IKE negotiated SAs cannot be removed through this routine, but are only removed by the system when their lifetime expires. After mutual agreement between the two nodes (establishing the SA for one-to-one communication), the lifetime is assigned to an IKE-negotiated SA. Refer to the [IPsec-IKE](#) chapter for further information.

The SA is immediately removed from the group even if it is being used actively. All the traffic based on the removed SA is dropped unless another SA with the same credentials already exists, or is added to the same group manually or is negotiated through IKE.

### Example

```
IPSEC_OUTBOUND_INDEX    sa_index;  
IPSEC_OUTBOUND_SA       outbound_sa;  
UINT32                  out_sa_index;  
CHAR                     group_name[] = "MY_GROUP";  
  
/* Set all the fields of outbound SA before adding it to any  
   IPsec  
   group. */  
.  
.
```

```
.  
  
/* Now create the outbound SA. "out_sa_index" contains the  
   index assigned to this SA. */  
if(IPSEC_Add_Outbound_Sa(group_name, &outbound_sa,  
                        &out_sa_index) != NU_SUCCESS)  
{  
    printf("ERROR: Unable to add the outbound SA. \n");  
}  
  
/* Set the index parameters. */  
sa_index.ipsec_group = group_name;  
sa_index.ipsec_index= out_sa_index;  
  
/* Remove the outbound SA now. */  
if(IPSEC_Remove_Outbound_SA(&sa_index) != NU_SUCCESS)  
{  
    printf("ERROR: Unable to remove the SA.\n");  
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Remove\_Policy

This function removes a policy from the Security Policy Database (SPDB) for the given IPsec group.

### Usage

```
STATUS IPSEC_Remove_Policy (CHAR    *group_name,  
                           UINT32  index);
```

### Arguments

- `group_name`  
A pointer to the name of the group from which to remove the policy.
- `index`  
The index of the policy to be removed.

### Return Values

- `NU_SUCCESS`  
The policy was successfully removed.
- `NU_TIMEOUT`  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- `IPSEC_INVALID_PARAMS`  
One of the input parameters is invalid.
- `IPSEC_NOT_FOUND`  
The group was not found or there is no policy corresponding to the passed index.

### Description

All the TCP and UDP ports in the networking stack associated with this policy are updated to not use this policy anymore, and the related fields are set to `NU_NULL`. However, their sessions and connections remain active. If more packets try to use these updated ports, then a search is made in the SPDB to find any other policy that allows the traffic to pass through. If one is found, the port is updated with the new policy and the associated traffic resumes on this port without any connection loss. Otherwise, no traffic is allowed to pass through the port and the port may get closed depending upon its kind and settings.

### Example

```
CHAR    group_name[] = "MY_GROUP";  
UINT8   src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};  
UINT8   dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};  
UINT32  policy_index;  
IPSEC_SELECTOR policy_selector;  
  
/* Initializing policy selector to look up policy index. */
```

```
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type   = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type     = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port  = IPSEC_WILDCARD;
policy_selector.ipsec_source_port      = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Call the remove policy routine. */
if(IPSEC_Remove_Policy(group_name, policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable to remove the policy.\n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Set\_Group\_Opt

This function sets the IPsec group value as specified by optname. The only option that can be set is “IPSEC\_DF\_PROCESSING”.

### Usage

```
STATUS IPSEC_Set_Group_Opt (CHAR *group_name,  
                           INT  optname,  
                           VOID *optval,  
                           INT  optlen);
```

### Arguments

- **group\_name**  
A pointer to the name of the group.
- **optname**  
The option to set. The only valid option is “IPSEC\_DF\_PROCESSING”. Only one of the following flags should be set:
  - IPSEC\_SET\_DF\_BIT**  
Set the DF bit in tunnel mode in the tunneling header.
  - IPSEC\_CLEAR\_DF\_BIT**  
Clear the DF bit in tunnel mode in the tunneling header.
  - IPSEC\_COPY\_DF\_BIT**  
Copy the DF bit from the tunneled header.
- **optval**  
A pointer to the new value to set. This value must be of UINT8 data type.
- **optlen**  
The total length, in bytes, of the optval parameter.

### Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.
- **IPSEC\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IPSEC\_NOT\_FOUND**  
The group was not found.

## Example

```
CHAR group_name[] = "MY_GROUP";
UINT8 df_setting ;

/* Set the group treatment of the DF bit to copy from the encapsulated
   header.*/
df_setting = IPSEC_COPY_DF_BIT;

/* Set the option. */
if(IPSEC_Set_Group_Opt(group_name, IPSEC_DF_PROCESSING,
                      &df_setting, sizeof(UINT8))!= NU_SUCCESS)
{
    printf("ERROR: Unable to set the option\n");
}

/* Set the group treatment of the DF bit to be cleared in the
   encapsulating header. */
df_setting = IPSEC_CLEAR_DF_BIT;

/* Set the option. */
if(IPSEC_Set_Group_Opt(group_name, IPSEC_DF_PROCESSING,
                      &df_setting, sizeof(UINT8))!= NU_SUCCESS)
{
    printf("ERROR: Unable to set the option \n");
}

/* Set the group treatment of DF bit to be set in the encapsulating
   header. */
df_setting = IPSEC_SET_DF_BIT;

/* Set the option. */
if(IPSEC_Set_Group_Opt(group_name, IPSEC_DF_PROCESSING,
                      &df_setting, sizeof(UINT8))!= NU_SUCCESS)
{
    printf("ERROR: Unable to set the option \n");
}
```

## Related Topics

[IPsec API Functions](#)

## IPSEC\_Set\_Policy\_Opt

This function sets the policy value as specified by optname.

### Usage

```
STATUS IPSEC_Set_Policy_Opt (CHAR    *group_name,  
                             UINT32  index,  
                             INT      optname,  
                             VOID     *optval,  
                             INT      optlen);
```

### Arguments

- group\_name

A pointer to the name of the group with which the policy is associated.

- index

The unique identifier for the policy.

- optname

The option to be set. Valid values are:

IPSEC\_SECURITY

Sets the policy security array. The optval parameter contains an [IPSEC\\_SECURITY\\_PROTOCOL](#) data structure.

IPSEC\_LIFETIME

Sets the lifetime for all the SAs that are established based on this policy. The optval parameter contains an [IPSEC\\_SA\\_LIFETIME](#) data structure. Valid only if IKE is included in the build.

IPSEC\_SA\_BUNDLE\_LIFETIME

Sets the lifetime associated with the policy bundle list. The optval parameter contains a UINT32 value.

IPSEC\_PFS\_GROUP\_DESC

Sets the PFS group associated with the policy. The optval parameter contains a UINT16 value. Valid only if IKE is included in the build.

- optval

A pointer to the new option value.

- optlen

The length, in bytes, of the value that is to be copied.

### Return Values

- NU\_SUCCESS

The request was successfully executed.

- **NU\_TIMEOUT**

The operation timed out. This is possibly because it was waiting for a semaphore that did not become available.

- **IPSEC\_INVALID\_PARAMS**

One of the input parameters is invalid.

- **IPSEC\_NOT\_FOUND**

The group was not found, or there is no policy corresponding to the passed index.

### Example – IPSEC\_SECURITY Option

```
INT    policy_security_size = sizeof(IPSEC_SECURITY_PROTOCOL);
CHAR    group_name[] = "MY_GROUP";
IPSEC_SECURITY_PROTOCOL policy_security;
UINT8    src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8    dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32    policy_index;
IPSEC_SELECTOR policy_selector;

/* Initializing the policy selector to look up the policy index. */
policy_selector.ipsec_source_type    = (IPSEC_SINGLE_IP |
                                         IPSEC_IPV4);
policy_selector.ipsec_dest_type      = (IPSEC_SINGLE_IP |
                                         IPSEC_IPV4);
policy_selector.selector_type        = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port = IPSEC_WILDCARD;
policy_selector.ipsec_source_port    = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Initialize policy_security parameters. */
policy_security.ipsec_protocol    = IPSEC_ESP;
policy_security.ipsec_security_mode = IPSEC_TRANSPORT_MODE;
policy_security.ipsec_auth_algo    = IPSEC_HMAC_MD5_96;
policy_security.ipsec_encryption_algo = IPSEC_DES_CBC;
policy_security.ipsec_flags        = IPSEC_IPV4;
policy_security.ipsec_sa_derivation = IPSEC_VALUE_FROM_POLICY;

/* Set the policy security with the given one. */
if(IPSEC_Set_Policy_Opt(group_name, policy_index, IPSEC_SECURITY,
                        (VOID*)&policy_security,
                        policy_security_size) != NU_SUCCESS)
{
    printf("ERROR: Unable to set IPSEC_SECURITY option.\n");
}
```



## Example – IPSEC\_LIFETIME Option

```
CHAR                group_name[] = "MY_GROUP";
INT                 lifetime_size = sizeof(IPSEC_SA_LIFETIME);
IPSEC_SA_LIFETIME   policy_sa_lifetime;
UINT8               src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8               dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32              policy_index;
IPSEC_SELECTOR      policy_selector;

/* Initializing policy selector to lookup policy index */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);
policy_selector.ipsec_dest_type = (IPSEC_SINGLE_IP |
                                   IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port = IPSEC_WILDCARD;
policy_selector.ipsec_source_port = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Initialize policy_sa_lifetime parameters. */
policy_sa_lifetime.ipsec_no_of_secs = 80;
policy_sa_lifetime.ipsec_expiry_action = IPSEC_REFRESH_SA;

/* Now set new lifetime options. */
if(IPSEC_Set_Policy_Opt(group_name, policy_index,
                        IPSEC_LIFETIME, (VOID*)&policy_sa_lifetime,
                        lifetime_size) != NU_SUCCESS)
{
    printf("ERROR: Unable to set IPSEC_LIFETIME option.\n");
}

Example - IPSEC_PFS_GROUP_DESC Option
CHAR                group_name[] = "MY_GROUP";
INT                 pfs_group_desc_size = sizeof(UINT16);
UINT16              policy_pfs_group_desc;
UINT8               src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8               dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32              policy_index;
IPSEC_SELECTOR      policy_selector;

/* Initializing the policy selector to look up the policy index. */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.ipsec_dest_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port = IPSEC_WILDCARD;
policy_selector.ipsec_source_port = IPSEC_WILDCARD;
```

```
memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Initialize the policy_pfs_group_desc parameter. */
policy_pfs_group_desc = IKE_GROUP_MODP_768;

/* Set it into the desired policy. */
if(IPSEC_Set_Policy_Opt(group_name, policy_index, IPSEC_PFS_GROUP_DESC,
                        (VOID*)&policy_pfs_group_desc,
                        pfs_group_desc_size) != NU_SUCCESS)
{
    printf("ERROR: Unable to set IPSEC_PFS_GROUP_DESC \ option.\n");
}
```

### Example – IPSEC\_PFS\_GROUP\_DESC Option

```
CHAR        group_name[] = "MY_GROUP";
INT          pfs_group_desc_size = sizeof(UINT16);
UINT16       policy_pfs_group_desc;
UINT8        src_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)10};
UINT8        dst_ip[] = {(CHAR)192, (CHAR)168, (CHAR)0, (CHAR)20};
UINT32       policy_index;
IPSEC_SELECTOR policy_selector;
.
/* Initializing the policy selector to look up the policy index */
policy_selector.ipsec_source_type = (IPSEC_SINGLE_IP |
                                     IPSEC_IPV4);

policy_selector.ipsec_dest_type = (IPSEC_SINGLE_IP | IPSEC_IPV4);
policy_selector.selector_type = IPSEC_OUTBOUND;
policy_selector.ipsec_transport_protocol = IPSEC_WILDCARD;
policy_selector.ipsec_destination_port = IPSEC_WILDCARD;
policy_selector.ipsec_source_port = IPSEC_WILDCARD;

memcpy(policy_selector.ipsec_source_ip.ipsec_addr, src_ip, IP_ADDR_LEN);
memcpy(policy_selector.ipsec_dest_ip.ipsec_addr, dst_ip, IP_ADDR_LEN);

/* Get the policy matching passed selector and type. */
if(IPSEC_Get_Policy_Index(group_name, &policy_selector,
                          IPSEC_OUTBOUND,
                          &policy_index) != NU_SUCCESS)
{
    printf("ERROR: Unable get the policy index.\n");
}

/* Initialize the policy_pfs_group_desc parameter. */
policy_pfs_group_desc = IKE_GROUP_MODP_768;

/* Set it into the desired policy. */
if(IPSEC_Set_Policy_Opt(group_name, policy_index, IPSEC_PFS_GROUP_DESC,
```

```
(VOID*)&policy_pfs_group_desc,  
pfs_group_desc_size) != NU_SUCCESS)  
{  
    printf("ERROR: Unable to set IPSEC_PFS_GROUP_DESC \ option.\n");  
}
```

## Related Topics

[IPsec API Functions](#)

# Customizations

This section describes the customization of Nucleus IPsec to add new authentication and encryption algorithms.

## Support for New Authentication Algorithms

Nucleus IPsec can be customized to add new authentication algorithms. The file *ipsec/ips\_cfg.c* defines the global variable `IPSEC_Authentication_Algos` (type is `IPSEC_AUTH_ALGO`), an array of supported authentication algorithms. [Table 7-15](#) describes the members of this structure:

**Table 7-15. Authentication Algorithms**

Members	Description
<code>ipsec_key_len</code>	(UINT8) The length of the key for this algorithm in bytes.
<code>ipsec_digest_len</code>	(UINT8) The length of the digest that is sent over the wire in bytes.
<code>ipsec_algo_identifier</code>	(UINT8) Identifier used to specify an authentication algorithm in Nucleus IPsec.

Note that these parameters are passed to Nucleus SSL to perform the actual processing of the algorithm. Therefore, the new algorithm must also be added to Nucleus SSL. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for instructions on adding algorithms to the Nucleus SSL library.

## IPSEC\_TOTAL\_AUTH\_ALGO

In the file *ipsec/ips\_cfg.h*, update the macro `IPSEC_TOTAL_AUTH_ALGO` to reflect the total number of authentication algorithms supported.

## IPSEC\_AUTH\_1\_TO\_1

If the index of the new entry in `IPSEC_Authentication_Algos` is equal to `ipsec_algo_identifier`, Nucleus IPsec does not search the array each time it authenticates a packet. Instead, it accesses the index directly, because there is a one-to-one mapping between the index and the identifier. To indicate that all algorithms have a one-to-one mapping between their index and identifier, set `IPSEC_AUTH_1_TO_1` in the file `ipsec/ips_cfg.h` to `NU_TRUE`. Note that by default this macro is set to `NU_TRUE`.

## Support for New Encryption Algorithms

Nucleus IPsec can be customized to add new encryption algorithms. The file `ipsrc/ips_cfg.c` defines the global data structure `IPSEC_Encryption_Algos` (type is `IPSEC_ENCRYPTION_ALGO`), an array of supported encryption algorithms. [Table 7-16](#) describes the members of this structure:

**Table 7-16. Encryption Algorithms**

Members	Description
<code>ipsec_key_len</code>	(UINT16) The length of the key for this algorithm in bytes.
<code>ipsec_algo_identifier</code>	(UINT8) Identifier used to specify an encryption algorithm in Nucleus IPsec.
<code>evp_cipher</code>	(void*) Pointer to hold the EVP structure corresponding to the encryption algorithm.

Note that these parameters are passed to Nucleus SSL to perform the actual processing of the algorithm. Therefore, the new algorithm must also be added to Nucleus SSL. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for instructions on adding algorithms to the Nucleus SSL library.

## IPSEC\_TOTAL\_ENCRYPT\_ALGO

In the file `ipsec/ips_cfg.h`, update the macro `IPSEC_TOTAL_ENCRYPT_ALGO` to reflect the total number of encryption algorithms supported.

## IPSEC\_ENCRYPT\_1\_TO\_1

If the index of the new entry in `IPSEC_Encryption_Algos` is equal to `ipsec_algo_identifier`, Nucleus does not search the array each time it encrypts a packet. Instead, it accesses the index directly, because there is a one-to-one mapping between the index and the identifier. To indicate that all algorithms have a one-to-one mapping between their index and identifier, set `IPSEC_ENCRYPT_1_TO_1` in the file `ipsec/ips_cfg.h` to `NU_TRUE`. Note that by default this macro is set to `NU_TRUE`.

# Nucleus IPsec Processing

The following discussion is for advanced users who want to better understand the internal Nucleus IPsec processing.

## Outbound Processing

When IPsec is included in the Nucleus NET build and enabled on some interface, all IP packets going through this interface are passed through the IPsec layer before being passed to the MAC layer. The following is the sequence of how a packet traverses through the networking stack.

1. Application Layer
2. Transport Layer
3. IP Layer
4. IPsec Layer

If an IPsec policy allows the packet to bypass IPsec processing on the outbound packet, then no security is applied. Otherwise, the packet is simply discarded, and an error is returned. If security must be applied, then it is encoded with the policy's securities, and the encoded packet is passed further down the stack.

- Back to the IP Layer
- MAC Layer

The following sections provide a high-level description of outbound processing.

## TCP Case

In the TCP case, an IPsec policy lookup is performed for the very first TCP packet during TCP session establishment. If some IPsec policy is found, it is cached in the corresponding TCP port structure so further communication using the same TCP port does not require a policy lookup for every outgoing TCP packet. However, in some cases a policy lookup could be required even during the TCP communication. When a policy is cached in the TCP port, the respective SA bundle is also cached so that it can be applied afterwards. SA bundles are used internally by Nucleus IPsec for applying the securities over the packets.

## UDP Case

In the UDP case, an IPsec policy lookup is performed, and if found, the policy as well as the packet's selector are cached in the UDP port structure. For subsequent packets, the packet's selector is matched against the cached one to check if the cached policy can be used in order to avoid policy lookup. If the packet's selector does not match the cached one, a policy lookup is performed. Upon successful lookup, the new policy is cached, replacing the old one. Similarly, this process continues for the rest of the life of UDP port.

## Policy Usage

Using the policy, it is determined whether the packet should be discarded, allowed to bypass IPsec, or whether IPsec must be applied. The following steps are for the case where IPsec is to be applied.

## Policy Security Array and Security Associations (SAs) Relationship

For each element in the security array (`ipsec_security`) of a policy, there must be a corresponding security association in the SADB. The element in `ipsec_security` indicates whether to use the selector derived from the packet or the one from the policy. After finding the selector from the previous step, an outbound SA bundle lookup is made against the policy's bundle list. If the SA bundle is not found, one is created and is added to the bundle list. The next time it is needed, the bundle is directly retrieved from the security policy. Finally, outbound SAs are searched for each of the elements in the `ipsec_security` array of the policy, and their indexes are stored in the outbound bundle.

---

### Note



If some outbound SA is not found in the outbound SADB, IKE is used to negotiate the SA.

---

---

### Note



`ipsec_sa_derivation` in the [Related Topics](#) data structure determines whether the selector from the packet or the policy should be used. It is more efficient to use the selector from the policy. This ensures that there is only one SA bundle per policy. Therefore, once the SA bundle has been associated with a policy, a search is not required to find an SA bundle. If the selector from the packet is used, it is possible to have more than one SA bundle per policy. A search is then be required to find the appropriate bundle.

---

IPsec is now applied according to the SA bundle found/created.

## Inbound Processing

When IPsec is enabled on an interface, all incoming IP packets must pass through the IPsec layer. The following is the sequence of how inbound processing is performed through the stack.

1. MAC Layer
2. IP Layer
3. IPsec Layer

If the incoming packet contains an IPsec header, an SA is searched to determine whether an agreement exists between the local and remote host. This agreement, if found, specifies exactly how the incoming packet is to be decoded. After the packet is decoded,

an IPsec policy is searched for it. The policy is used to verify whether the incoming packet provides the required level of security. If the policy requirements are satisfied, the packet is passed to the IP and higher protocol layers.

4. Back to the IP Layer
5. Transport Layer
6. Application Layer

The following is a high-level description of inbound processing:

- The SPI, destination address, and upper-layer protocol are used to find an SA that is to be applied to the packet. If found, the packet is decoded with the SA.

---

**Note**



If an SA is not found, the packet is dropped.

---

- These steps are applied until all IPsec headers have been processed on the packet.
- If the higher-layer protocol is neither TCP nor UDP, then a selector is created from the IP packet. This selector is used to find a security policy. The security policy indicates whether the packet should be discarded, allowed to bypass IPsec, or whether IPsec should be applied.
- If the received packet is TCP, the TCP port contains a pointer to the policy that is applicable. If no policy is associated, a policy lookup is made using the packet selectors. If found, the policy is cached in the TCP port structure for future use.
- If the received packet is UDP, then the packet selector is checked with the selector in the UDP port structure to determine whether the last packet was received from the same node. If it was, the cached policy is used. Otherwise, the SPDB is searched and the new policy is cached in the UDP port structure. The new packet selector is also cached in the UDP port structure.

---

**Note**



If a policy is not found, the packet is discarded. Also note that when finding a policy, the list is searched starting from the beginning. If multiple policies match the selection criteria, the policy that occurs the earliest in the list is selected.

---

- Using the policy, it is determined whether the SA application was correct. The total number of SAs applied must be equal to the size of the security array in the respective policy.

If the application of IPsec was according to the policy, the packet is passed to the upper-layer protocol; otherwise, it is discarded.


















































































































The IPsec protocol requires establishment of Security Associations (SA) for securing IP-based communication. An SA defines an agreement between two nodes on the cipher algorithm, hash algorithm, and keys to be used for encrypting and authenticating communication data. For two nodes to communicate, the SA parameters must be identical at both ends. Furthermore, continuous use of the same algorithm keys introduces security issues. Keys must be changed periodically. It is almost impossible for a System Administrator to manually manage Security Associations, especially in large networks.

Internet Key Exchange (IKE) is a protocol designed to automate management of SA. It dynamically negotiates, establishes, and updates IPsec SA. The second version of this protocol is known as IKEv2. Nucleus supports both the IKEv1 and IKEv2 protocols.

---

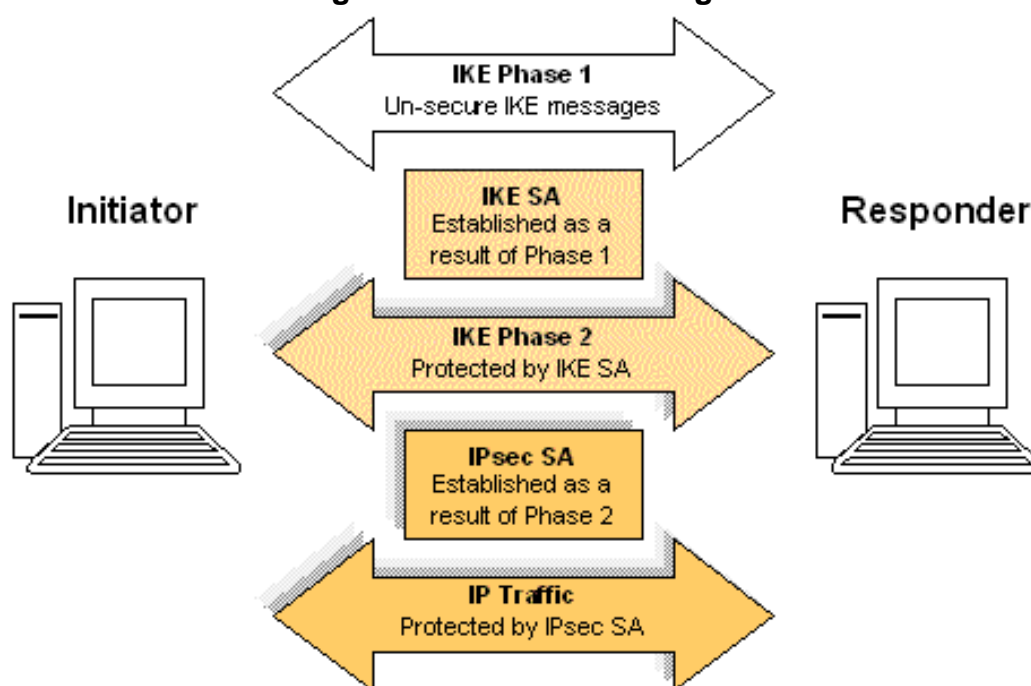
 **Warning** In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## IKEv1 Exchange Overview

The process of negotiating an SA is called an IKE exchange. There are always two peers involved in an IKE exchange. The peer that sends the first message of an exchange is referred to as the Initiator and the other peer is the Responder.

**Figure 8-1. IKEv1 Exchange**



An IKE exchange takes place in two phases as follows:

## Phase 1

When two peers wish to communicate, it is assumed that no secure channel exists between them. The purpose of Phase 1 is to secure the communication channel by authenticating the negotiating parties and then establishing an IKE Security Association (IKE SA). The negotiated IKE SA is used to protect Phase 2 messages.

Phase 1 could be performed in one of two possible modes: Main Mode and Aggressive Mode. The final result of both modes is the same. Aggressive Mode is faster than Main Mode, although it is also slightly less secure. Several different methods have been defined for authentication of the negotiating peers. The following are the most commonly used authentication methods:

- Authentication using Pre-shared Keys – As the name suggests, an authentication key must be shared out-of-band, between the two peers. The mechanism used to establish the shared key is out of the scope of this document.
- Authentication using Digital Signatures – Digital signature algorithms, such as RSA, are used to authenticate the peers. This requires setting up certificates on both ends.

## Phase 2

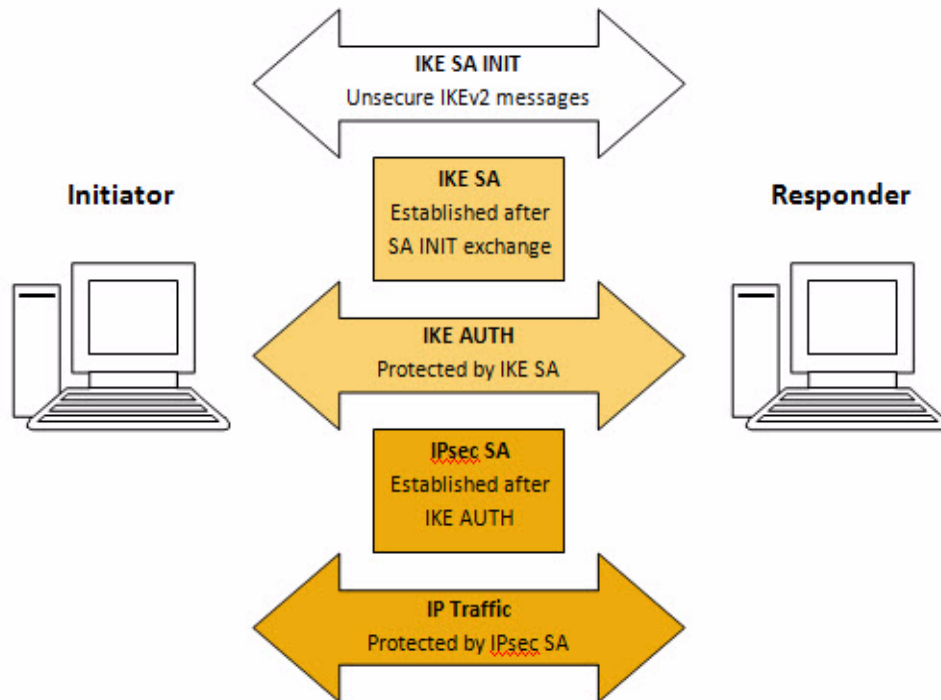
Phase 2, also called Quick Mode, is used to negotiate IPsec SAs. All messages exchanged in Phase 2 are protected using the IKE SA negotiated in Phase 1. Multiple Phase 2 exchanges can

be performed after a single Phase 1. This results in quicker exchanges because Phase 1 is computationally more intensive than Phase 2.

## IKEv2 Exchange Overview

An IKEv2 exchange requires only four packets being exchanged between peers. A brief description of various stages of exchange is shown in [Figure 8-2](#):

**Figure 8-2. IKEv2 Exchange**



### IKE\_SA\_INIT

This exchange is analogous to IKE Phase 1 in version 1. This pair of messages is not encrypted. Both end points exchange suits that are acceptable to them. This exchange results in establishment of IKE SA. Information exchanged in this phase includes sets of algorithms acceptable for each node, key exchange material and nonce values.

### IKE\_AUTH

An IKE\_AUTH exchange is carried out authenticating the IKE SA established in the previous exchange. Both sides present to each other their identification and try to prove it. Authentication can be done in one of two ways as described in [Phase 1](#) under [IKEv1 Exchange Overview](#).

An IPsec SA (termed as a child SA in IKEv2) is also established in the same IKE\_AUTH exchange.

## CREATE\_CHILD\_SA

Once an IKE SA is established, additional child SAs can be created using CREATE\_CHILD\_SA exchanges under the same IKE SA. The CREATE\_CHILD\_SA exchange is used to establish an IPsec SA. This is usually required when a previously established IPsec SA expires and a new one must replace it.

---

### Note



This document is not a reference on the IKE or IPsec protocols. Protocol concepts are briefly discussed where necessary for clarification. Readers are referred to the large amount of literature available on this subject. *The New Security Standard for the Internet, Intranets, and Virtual Private Networks* by Nagan and Doraswamy and Dan Harkins (ISBN 0-13-011898-2) is one recommended reading.

---

## Nucleus IPsec - IKE Protocol

Nucleus IPsec includes an implementation of the IKEv1 and IKEv2 protocols. You can utilize this feature to dynamically negotiate SAs for Nucleus IPsec.

## Supported RFCs

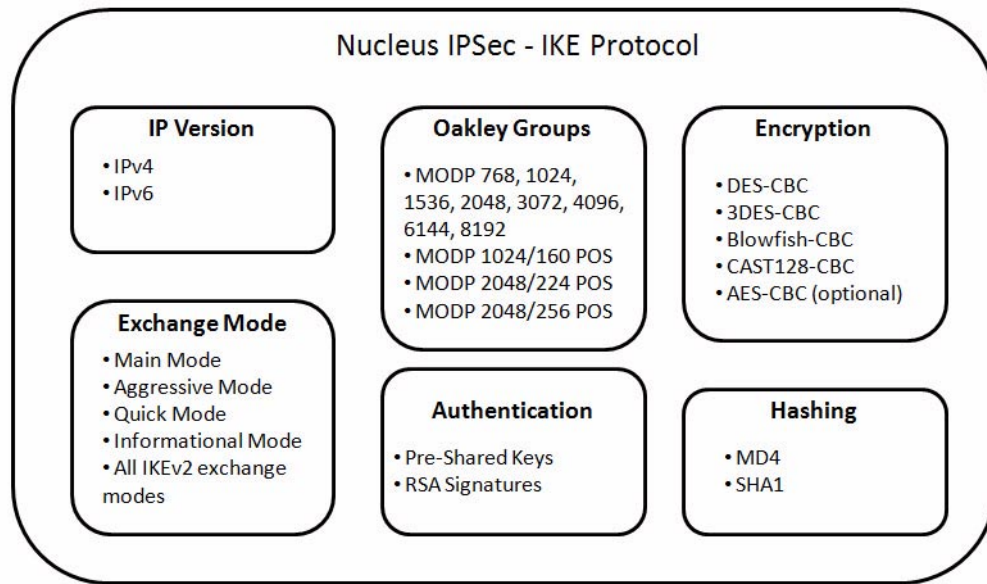
The following is the list of RFCs supported by Nucleus IPsec - IKE implementation.

- RFC 2407 – The Internet IP Security Domain of Interpretation for ISAKMP
- RFC 2408 – Internet Security Association and Key Management Protocol (ISAKMP)
- RFC 2409 – The Internet Key Exchange (IKE)
- RFC 3526 – More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)
- RFC 2459 – Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC 4306 - Internet Key Exchange (IKEv2) Protocol
- RFC 4307 - Cryptographic Algorithms for Use in the Internet Key Exchange Version 2
- RFC 4718 - IKEv2 Clarifications and Implementation Guidelines
- RFC 4809 – Requirements for an IPsec Certificate Management Profile (Partially supported)

## Supported Algorithms and Protocols

IKE supports both IPv4 and IPv6 protocols.

**Figure 8-3. Nucleus IPsec - IKE Components**



## Exchange Modes for IKEv1

The following are the exchange modes supported in Phase 1 and Phase 2 of IKE:

- Main Mode
- Aggressive Mode
- Quick Mode
- Informational Mode (one-way transmission of information helpful for managing SAs)

## IKEv2 Exchanges

The following exchange types are supported for IKEv2:

- IKE\_SA\_INIT
- IKE\_AUTH
- IKE\_CREATE\_CHILD

## Hashing Algorithms

The following are the hash algorithms that can be negotiated for IKE SAs. These hash algorithms are used to authenticate IKE messages.

- MD5

- SHA1

## Encryption Algorithms

The following are the encryption algorithms that can be negotiated for IKE SAs. These encryption algorithms are used to encrypt IKE messages.

- DES-CBC
- 3DES-CBC
- Blowfish-CBC
- CAST128-CBC
- AES-CBC

## Pseudo Random Functions

The following PRF functions are supported for IKEv2:

- HMAC-MD5
- HMAC-SHA1

## Authentication Methods

The following are the authentication methods that can be used in Phase 1 of IKE:

- Authentication using Pre-shared Keys
- Authentication using Signatures

## Oakley Groups

Oakley groups are used during Phase 1 to calculate keys for the IKE SA. IKE supports the following Oakley groups. The second group is more secure than the first.

- First Oakley Group (MODP 768)
- Second Oakley Group (MODP 1024)
- Fifth Oakley Group (MODP 1536)
- MODP 2048
- MODP 3072
- MODP 4096
- MODP 6144

- MODP 8192
- MODP 1024 with 160 bit Prime Order Sub-group
- MODP 2048 with 224 bit Prime Order Sub-group
- MODP 2048 with 256 bit Prime Order Sub-group

## Configurable Macros for IKEv1

All configurable parameters for IKE are located in *os/include/networking/ike\_cfg.h* and *os/networking/ike/ike\_cfg.c*. Some of these macros are also used by the IKEv2 code. This section describes some of these parameters. Advanced configuration options are discussed in [“IKE Customizations”](#) on page 1414.

### IKE\_TIMEOUT

The number of timer ticks IKE waits on a resource before timing out. Valid values are from 1 to 4,294,967,293. This can also be set to `NU_SUSPEND` for indefinite suspension. The default is `NU_SUSPEND`.

### IKE\_ENABLE\_BLOCKING\_REQUEST

Nucleus IPsec always invokes IKE with a non-blocking request. It does not wait for an IKE exchange to complete. However, IKE supports blocking requests when explicitly invoked by a user application. This allows the application to wait until an IKE exchange completes or fails. Setting this macro to `NU_TRUE` enables support for blocking IKE requests. Setting this macro to `NU_FALSE` disables support for blocking IKE requests. The default is `NU_FALSE`.

### IKE\_MAX\_WAIT\_EVENTS

If blocking requests are enabled in IKE, then multiple user application tasks can block the completion or failure of an IKE exchange. This macro defines the maximum number of such requests. It must be set to a value from 1 to 31. The default is 5.

### IKE\_DEBUG

Setting this macro to `NU_TRUE` enables debug mode. In debug mode, extensive error checking and error logging is performed. Debug mode is helpful in tracing problems in IKE configurations. Setting this macro to `NU_FALSE` disables debug mode. The default is `NU_TRUE`.

## IKE\_DEBUG\_LOG

When debugging is enabled, this macro is used to output debug messages. The default logging is performed using the Nucleus NET NLOG component. However, this macro could be set to a target specific output function, which takes a null-terminated string as a parameter and displays it. This could be a function similar to printf. The default is NLOG\_Error\_Log((str), NERR\_INFORMATIONAL, \_\_FILE\_\_, \_\_LINE\_\_).

## IKE\_INCLUDE\_VERSION\_2

Setting this NU\_TRUE enables support for IKEv2 protocol. When IKEv2 is enabled, but version is not specified in IKE Policy flags, the version defaults to 1.

## IKE\_INCLUDE\_MAIN\_MODE

Setting this macro to NU\_TRUE enables support for Main Mode in IKE Phase 1. Setting this macro to NU\_FALSE, disables support for Main Mode. Note that at least one Phase 1 Exchange Mode must be enabled. The default is NU\_TRUE.

## IKE\_INCLUDE\_AGGR\_MODE

Setting this macro to NU\_TRUE enables support for Aggressive Mode in IKE Phase 1. Setting this macro to NU\_FALSE disables support for Aggressive Mode. Note that at least one Phase 1 Exchange Mode must be enabled. The default is NU\_FALSE.

## IKE\_INCLUDE\_INFO\_MODE

Informational Exchange is a one-way transmission of information that can be used for SA management. Setting this macro to NU\_TRUE enables support for this exchange. Setting this macro to NU\_FALSE disables support for this exchange. The default is NU\_TRUE.

## IKE\_INCLUDE\_PSK\_AUTH

Setting this macro to NU\_TRUE enables support for IKE authentication using a pre-shared key. Setting this macro to NU\_FALSE disables this support. At least one IKE authentication method must be enabled. The default is NU\_TRUE.

## IKE\_INCLUDE\_SIG\_AUTH

Setting this macro to NU\_TRUE enables support for IKE authentication using RSA signatures. Setting this macro to NU\_FALSE disables this support. At least one IKE authentication method must be enabled. The default is NU\_FALSE.



## IKE\_ENABLE\_INITIAL\_CONTACT

INITIAL-CONTACT is an IKE notification used to synchronize SAs when a node is unexpectedly restarted. Informational Mode must also be enabled for this support. Setting this macro to `NU_TRUE` enables the INITIAL-CONTACT notification. Setting this macro to `NU_FALSE` disables INITIAL-CONTACT notification. The default is `NU_TRUE`.

## IKE\_ENABLE\_DOMAIN\_NAME\_ID

Domain name support in identification is used during IKE authentication for looking up pre-shared keys. Setting this macro to `NU_TRUE` enables support for domain names (FDQN) and user domain names (USER\_FDQN) in IKE. Setting this macro to `NU_FALSE` disables support for domain names. If this support is disabled, only single IPv4 and IPv6 addresses can be specified as the local and remote identities in IKE policies and pre-shared keys. The default is `NU_FALSE`.

## IKE\_RETAIN\_LAST\_MESSAGE

In both Phase 1 and Phase 2 exchanges, the last message of the exchange can be lost during transmission. Setting this macro to `NU_TRUE` allows retention of the last message of the exchange so that it can be retransmitted if it is lost. The message is retained until the exchange times out. Note that enabling this option consumes more memory and is not recommended if a large number of IKE exchanges will be performed simultaneously. The memory overhead involves maintaining a buffer of length `IKE_MAX_OUTBOUND_PACKET_LEN` for as long as each IKE exchange does not timeout, even after successful completion. This macro applies to the last message of an exchange only. All other messages are always retained until a reply is received. Setting this macro to `NU_FALSE` disables this feature. The default is `NU_FALSE`.

## IKE\_PHASE1\_DEFAULT\_XCHG

If both Main Mode and Aggressive Mode are allowed by an IKE policy, then this macro specifies the preferred mode of exchange. Valid values for this macro are:

- `IKE_XCHG_MAIN`
- `IKE_XCHG_AGGR`

The default is `IKE_XCHG_MAIN`.

## IKE\_INCLUDE\_MODP\_1024

Setting this macro to `NU_TRUE` enables support for the 1024-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_TRUE`.

## **IKE\_INCLUDE\_MODP\_1536**

Setting this macro to `NU_TRUE` enables support for the 1536-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_2048**

Setting this macro to `NU_TRUE` enables support for the 2048-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_3072**

Setting this macro to `NU_TRUE` enables support for the 3072-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_4096**

Setting this macro to `NU_TRUE` enables support for the 4096-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_6144**

Setting this macro to `NU_TRUE` enables support for the 6144-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_8192**

Setting this macro to `NU_TRUE` enables support for the 8192-bit Modular Exponential (MODP) used by IKE to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_1024\_160\_POS**

Setting this macro to `NU_TRUE` enables support for the 1024-bit MODP with 160-bit Prime Order Sub-group which is used specifically by IKEv2 to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_2048\_224\_POS**

Setting this macro to `NU_TRUE` enables support for the 2048-bit MODP with 224-bit Prime Order Sub-group which is used specifically by IKEv2 to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MODP\_2048\_256\_POS**

Setting this macro to `NU_TRUE` enables support for the 2048-bit MODP with 256-bit Prime Order Sub-group which is used specifically by IKEv2 to negotiate a shared secret. Setting this macro to `NU_FALSE` disables support for this MODP. The default is `NU_FALSE`.

## **IKE\_SOFT\_LIFETIME\_OFFSET**

This macro defines the offset between an IPsec SA's soft and hard lifetimes in seconds. The soft lifetime expires this number of seconds before the hard lifetime. If the hard lifetime is less than or equal to this value, then the soft lifetime is not set. Valid values are from 1 to 28800. The default is 10.

## **IKE\_RESEND\_INTERVAL**

IKE uses the UDP protocol for message transmission. Therefore, it has to handle message loss and retransmission. This macro defines the number of seconds to wait before the first retransmission of a message. This interval is applied with an exponential back-off to all subsequent retransmissions. Valid values are from 1 to 28800. The default is 2.

## **IKE\_RESEND\_COUNT**

This macro defines the maximum number of message retransmissions before giving up on an exchange. Valid values are from 1 to 10. The default is 5.

## **IKE\_PHASE1\_TIMEOUT**

This macro defines the time, in seconds, within which an IKE Phase 1 exchange must complete. If this timeout is exceeded, the Phase 1 exchange is aborted. Valid values are from 1 to 28800. The default is 60.

## **IKE\_PHASE2\_TIMEOUT**

This macro defines the time, in seconds, within which an IKE Phase 2 exchange must complete. If this timeout is exceeded, the Phase 2 exchange is aborted. Valid values are from 1 to 28800. The default is 45.

## IKE\_MAX\_DELETE\_SPI

This macro defines the maximum number of SAs that can be specified for deletion in a single Delete payload. This payload is normally sent by a remote host to synchronize SAs. Each SA that is to be deleted is identified by a unique Security Parameter Index (SPI). Valid values are from 1 to 20. The default is 2.

## IKE\_MAX\_PROPOSALS

This macro defines the maximum number of Proposal Payloads allowed in an SA Payload. This is always 1 for a Phase 1 exchange and is equal to the Security size in the Nucleus IPsec Policy, for Phase 2. For example, if AH and ESP are used, then the maximum number of Proposal Payloads in an IKE negotiation would be 2. Valid values are from 1 to 50. The default is 4.

## IKE\_MAX\_TRANSFORMS

This macro defines the maximum number of Transform Payloads in a Proposal Payload. This is the number of alternative attribute groups available in a Phase 1 exchange. For a Phase 2 exchange, it is the product of all possibilities of the phase 2 Security Protocol. For example, an ESP proposal that specifies “(DES OR 3DES encryption) AND (MD5 OR SHA1 authentication)” would contain  $2 \times 2 = 4$  Transform Payloads. Valid values are from 1 to 50. The default is 5.

## IKE\_MAX\_DOMAIN\_NAME\_LEN

This macro defines the maximum length of a domain name, including the null terminator. This is only used if `IKE_ENABLE_DOMAIN_NAME_ID` is set to `NU_TRUE`. The length must be a multiple of 4 bytes. Valid values must lie within a range of 4 to 256. The default is 56.

## IKE\_MAX\_GROUP\_NAME\_LEN

This macro defines the maximum length of an IKE Group name, including the null terminator. This length must be a multiple of 4 bytes. Valid values must lie within a range of 4 to 32. The default is 20.

## IKE\_OUTBOUND\_NONCE\_DATA\_LEN

This macro defines the length of Nonce data, in bytes, sent during Phase 1 and Phase 2 exchanges. Valid values are from 4 to 252. The default is 20.

## IKE\_MAX\_INBOUND\_PACKET\_LEN

This macro defines the maximum length of an inbound IKE packet, in bytes. Valid values are from 200 to the maximum UDP packet size supported by the network interface being used to transmit IKE messages. This must not exceed 65535 in any case. The default is 1400.

## IKE\_MAX\_OUTBOUND\_PACKET\_LEN

This macro defines the maximum length of an outbound IKE packet, in bytes. Valid values are from 200 to the maximum UDP packet size supported by the network interface being used to transmit IKE messages. This must not exceed 65535 in any case. The default is 512.

## IKE\_MAX\_BUFFERS

This macro defines the total number of outbound IKE buffers allocated when IKE is initialized. The maximum number of buffers needed by IKE is approximately equal to the maximum number of simultaneous IKE exchanges. The memory pool passed to the initialization function must be large enough to allocate these buffers. The default is 10.

## IKE\_TASK\_STACK\_SIZE

The IKE task receives and processes all incoming exchange messages. It carries out the core tasks of the IKE service. This macro defines the stack size of the IKE task, in bytes. Maximum stack usage may be affected by other configuration macros such as `IKE_MAX_DELETE_SPI`, `IKE_MAX_PROPOSALS`, `IKE_MAX_TRANSFORMS` and `IKE_MAX_DOMAIN_NAME_LEN`. A value of 2500 or greater is recommended. The default is 3000.

## IKE\_TASK\_PRIORITY

This macro defines the IKE task priority. It may vary from 0 to 255. The lower the numeric value, the higher the task's priority. The default is 3.

## IKE\_TASK\_TIME\_SLICE

This macro defines the time slice for the IKE task. It indicates the maximum number of timer ticks that may expire while executing the task. A value of 0 disables time slicing. The default is 0.

## IKE\_TASK\_PREEMPT

This macro defines preemption for the IKE task. Setting this to `NU_PREEMPT` enables preemption and setting this to `NU_NO_PREEMPT` disables preemption. Disabling preemption also disables time slicing. The default is `NU_PREEMPT`.

## IKE\_EVENT\_TASK\_STACK\_SIZE

The IKE event task is responsible for handling events such as IKE SA timeouts and Phase 1 and 2 timeouts. It also services IKE exchange requests made by Nucleus IPsec. This macro defines the stack size of the IKE event task, in bytes. Maximum stack usage may be affected by other configuration macros such as `IKE_MAX_DELETE_SPI`, `IKE_MAX_PROPOSALS`, `IKE_MAX_TRANSFORMS` and `IKE_MAX_DOMAIN_NAME_LEN`. A value of 2500 or greater is recommended. The default is 2500.

## IKE\_EVENT\_TASK\_PRIORITY

This macro defines the IKE event task priority. It may vary from 0 to 255. The lower the numeric value, the higher the task's priority. The default priority is set to the same as that of the Nucleus NET tasks. The default is 3.

## IKE\_EVENT\_TASK\_TIME\_SLICE

This macro defines the time slice for the IKE event task. It indicates the maximum number of timer ticks that may expire while executing the task. The default value of 0 disables time slicing.

## IKE\_EVENT\_TASK\_PREEMPT

This macro defines preemption for the IKE event task. Setting this to `NU_PREEMPT` enables preemption and setting this to `NU_NO_PREEMPT` disables preemption. Disabling preemption also disables time slicing. The default is `NU_PREEMPT`.

## IKE\_INCLUDE\_DES

Setting this macro to `NU_TRUE` enables support for DES encryption in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_TRUE`.

## IKE\_INCLUDE\_3DES

Setting this macro to `NU_TRUE` enables support for 3DES encryption in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_TRUE`.

## IKE\_INCLUDE\_BLOWFISH

Setting this macro to `NU_TRUE` enables support for Blowfish encryption in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase

2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_TRUE`.

## **IKE\_INCLUDE\_CAST128**

Setting this macro to `NU_TRUE` enables support for CAST-128 encryption in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_TRUE`.

## **IKE\_INCLUDE\_AES**

Setting this macro to `NU_TRUE` enables support for AES encryption in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_MD5**

Setting this macro to `NU_TRUE` enables support for MD5 Authentication (derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm) in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. MD5 must always be enabled in the Nucleus SSL library even if this macro is set to `NU_FALSE`. This is required because IKE internally uses MD5 in addition to Phase 1 authentication. The default is `NU_TRUE`.

## **IKE\_INCLUDE\_SHA1**

Setting this macro to `NU_TRUE` enables support for SHA1 Authentication in Phase 1. Setting this macro to `NU_FALSE` disables this support. The ability to negotiate this algorithm in Phase 2 is not affected by this macro. The Nucleus IPsec configuration dictates Phase 2 capabilities. The default is `NU_TRUE`.

## **IKE\_INCLUDE\_RSA**

Setting this macro to `NU_TRUE` enables support for RSA Signature-based authentication in Phase 1. Setting this macro to `NU_FALSE` disables this support. The default is `NU_FALSE`.

## **IKE\_INCLUDE\_PEM**

Setting this macro to `NU_TRUE` enables the support for using Base64 encoded certificate and private key files to be used for authentication with RSA Signature.

## IKE\_INCLUDE\_CRL\_SUPPORT

Setting this macro to `NU_TRUE` enables the support to check the certificate against a Certificate Revocation List (CRL). A CRL is a list of certificates issued by a certification authority that the authority no longer recommends as trusted. This verification against CRL is done only when the `IKE_VERIFY_CERT` flag is set in the policy options flags.

## IKE\_MAX\_ENCRYPT\_BLOCK\_LEN

This macro defines the maximum block length from all supported encryption algorithms. This need not be modified unless custom algorithms are added to IKE. See [“IKE Customizations”](#) on page 1414 for more information. The value of this macro must be a multiple of 4 bytes. The default is 8.

## IKE\_MAX\_HASH\_DATA\_LEN

This macro defines the maximum digest length from all supported hash algorithms. This need not be modified unless custom algorithms are added to IKE. See the [“IKE Customizations”](#) on page 1414 for more information. The value of this macro must be a multiple of 4 bytes. The default is 20 if `IKE_INCLUDE_SHA1` is set to `NU_TRUE` and 16 if `IKE_INCLUDE_SHA1` is set to `NU_FALSE`.

## Compile Time Options for IKEv2

The compile time configurations for version two are in the file *os/include/networking/ike\_cfg.h*.

## IKE2\_SA\_TIMEOUT

If IKE SA re-keying is enabled, wait this much time before re-keying. Unlike IKEv1, SA lifetimes are not negotiated in IKEv2, thus it needs to be configured here. The value defined here is used as the default value when timeout is not specified in the IKE policy.

## IKE2\_ENABLE\_NAT\_TRAVERSAL

Setting this to true enables the support negotiation of NAT traversal. (Since IPsec does not allow NAT traversal, this feature in IKEv2 will not be used).

## IKE2\_MAX\_PACKET\_LEN

IKEv2 RFC states that IKEv2 implementations should be able to handle packets up to 3000 bytes in length. This macro reflects this length.



## IKE2\_MESSAGE\_REPLY\_TIMEOUT

If you sent a request and did not receive the response in time, you need to resend the request. This specifies the time to wait before retransmission.

## IKE Database

In most cases, securing network communication using IPsec requires only two steps. The first step is to configure IPsec policies. Refer to the [IPsec](#) chapter for instructions on configuring IPsec policies. The second step is to configure IKE policies. This section explains the IKE database in which IKE policies are stored. An understanding of the IKE database is critical for the correct setup of IKE policies.

The most basic component of the IKE database is an IKE Group. One or more network interfaces can be added to a group. A group also contains IKE policies. These policies apply to all network interfaces associated with the parent IKE Group. Another component of the IKE database is the list of pre-shared keys. This list is referenced during an IKE exchange to authenticate remote hosts.

---

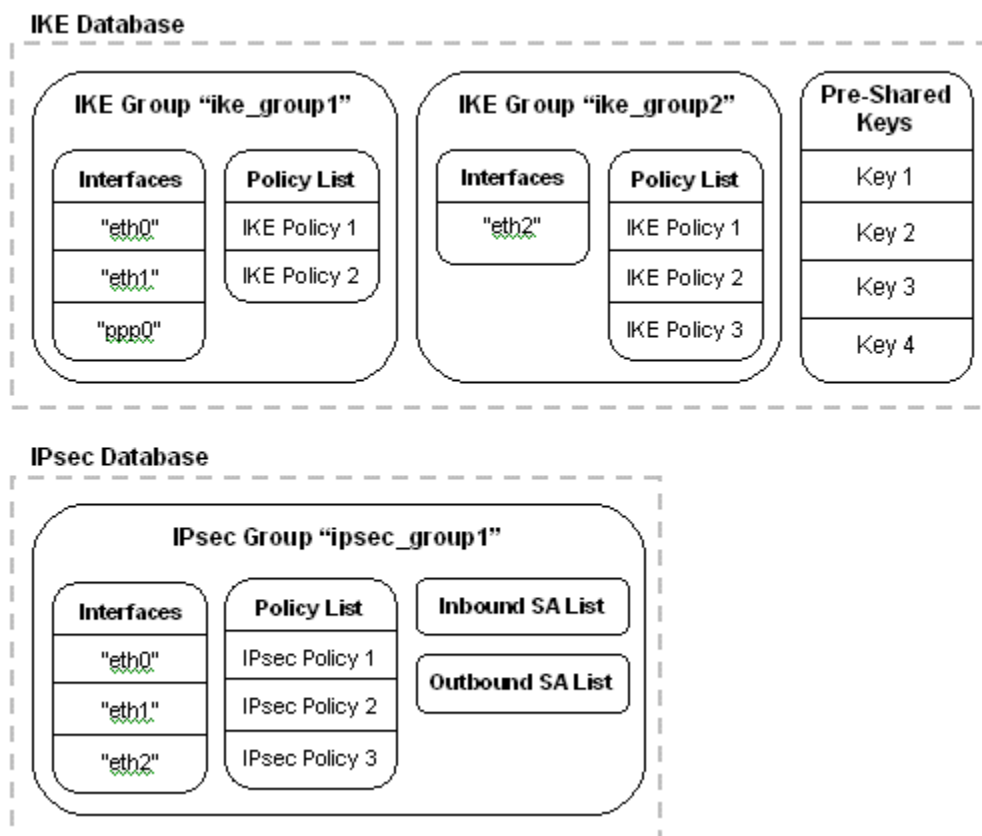
**Note**

In IKEv2 a separate "Pre-Shared Keys" database is not maintained. IKEv2 pre-shared keys are stored within the IKE policy.

---

Figure 8-4 shows an example of an IKE and IPsec database configuration.

**Figure 8-4. IKE and IPsec Database Configuration**



For example, when the database shown in Figure 8-4 is used, an IKE exchange can be performed on network interface "eth2", using "IKE Policy 3" and pre-shared key "Key 2". "IPsec Policy 3" can be the IPsec policy for which IPsec SAs are negotiated. Once the exchange completes, the resulting IPsec SAs are added to the Outbound and Inbound SA list in the IPsec group.

## IKE Policy

An IKE policy determines how key exchanges are carried out with a particular IKE node. In IKE, exchanges are performed in two phases. In the first phase, IKE negotiates an IKE SA that is used for protecting Phase 2 communication. In the second phase, the actual IPsec SAs are negotiated according to the security requirements of a Nucleus IPsec policy.

For example, when negotiating IPsec SAs, the IKE nodes may agree on using DES-CBC for encryption, MD5 for hashing and pre-shared keys for authentication. This negotiation is part of Phase 1. The keys exchanged for encryption and hashing are used to protect communication in the second phase. IPsec SAs are negotiated in Phase 2.

IKE maintains a list of policies. Each policy in a list is uniquely identified by an unsigned 32-bit integer and specifies a single IP address, a range of IP addresses, a subnet or a wildcard that matches all addresses. Policies in this list are stored in the same order by which they were added to the database. For example, new policies are always appended to the end of the policy list.

Table 8-1 gives an example of a policy list:

**Table 8-1. IKE Policy List**

Type	Value
Single IP Address	192.168.0.1
Range of IP Addresses	192.168.0.20 to 192.168.0.30
Range of IP Addresses	192.168.0.1 to 192.168.0.99
Single IP Address	192.168.0.2
Subnet	192.168.0.0 Subnet Mask 255.255.0.0
Wildcard	Value not applicable. Matches all addresses.

The list is searched starting from the beginning. If multiple policies match the selection criteria, the policy that occurs earliest in the list is selected. In the previous example, an IKE node with IP address 192.168.0.1 matches the first, third, fifth, and sixth policies. However, the first policy is used during negotiation since it is the first match found.

Note that the wildcard policy is usually the last policy in the list. It matches all addresses, so any policy which follows it is not matched.

## IKE Data Structures

The following section gives a description of the data structures that are used in IKE API functions related to policies. API functions are described in [IKE API Functions](#).

---

**Note**

The same API functions and data structures are used to define either IKEv1 or IKEv2 policies. However, some APIs or data structure members might be IKE version specific. Such version specific differences are explicitly mentioned in this document.

---

---

**Note**

Some members of the following data structures have been omitted intentionally. These are used internally by IKE.

---

### Note



At least one Nucleus IPsec policy that applies security must be present to allow an IKE exchange. Refer to the [IPsec](#) chapter for more information on creating Nucleus IPsec policies.

---

The following data structures are described:

- [IKE\\_POLICY](#)
- [IKE\\_POLICY\\_SELECTOR/IKE\\_IDENTIFIER](#)
- [IKE\\_SELECTOR\\_IP](#)
- [IKE\\_IPS\\_ID](#)
- [IKE\\_ATTRIB](#)
- [IKE\\_SA\\_LIFETIME](#)
- [IKE\\_KEY\\_PAIR](#)

## IKE\_POLICY

IKE\_POLICY is the typedef for the ike\_policy data structure. This structure is used to define IKEv1 and IKEv2 policies. The following is a description of this structure. Some of the members may have been omitted as they are used internally.

```
typedef struct ike_policy
{
    IKE_POLICY_SELECTOR ike_select;
    IKE_IDENTIFIER      ike_my_id;
    IKE_IDENTIFIER      ike_peers_id;
    IKE\_IPS\_ID           *ike_ids;
    IKE\_ATTRIB          *ike_xchg1_attribs;
    UINT8               ike_phase1_xchg;
    UINT8               ike_ids_no;
    UINT8               ike_xchg1_attribs_no;
    UINT8               ike_flags;
    #if (IKE_INCLUDE_VERSION_2 == NU_TRUE)
    UINT8               ike_version;
    UINT8               ike2_flags;
    UINT32              ike2_sa_timeout;
    #else
    UINT8               ike_pad2[3];
    #endif
} IKE_POLICY;
```

The members of this structure are defined in [Table 8-2](#). Some of the members were omitted as they are used internally.

**Table 8-2. IKE\_POLICY**

Member	Description
ike_select	<p>(IKE_POLICY_SELECTOR) The selector used to match an IKE node to this policy. This structure has been defined below. Only the following address types are allowed in this data member:</p> <ul style="list-style-type: none"><li>• Match all addresses (IKE_WILDCARD).</li><li>• Single IPv4 address (IKE_IPV4). ike_addr.iike_ip.iike_addr1 contains the IP address.</li><li>• Range of IPv4 addresses (IKE_IPV4_RANGE). IP addresses between ike_addr.iike_ip.iike_addr1 and ike_addr.iike_ip.iike_ext_addr.iike_addr2 inclusive.</li><li>• IPv4 subnet (IKE_IPV4_SUBNET). ike_addr.iike_ip.iike_addr1 contains the subnet address and ike_addr.iike_ip.iike_ext_addr.iike_addr2 contains the subnet mask.</li><li>• Single IPv6 address (IKE_IPV6). ike_addr.iike_ip.iike_addr1 contains the IP address.</li><li>• Range of IPv6 addresses (IKE_IPV6_RANGE). IP addresses between ike_addr.iike_ip.iike_addr1 and ike_addr.iike_ip.iike_ext_addr.iike_addr2 inclusive.</li><li>• IPv6 subnet (IKE_IPV6_SUBNET). ike_addr.iike_ip.iike_addr1 contains the subnet address and ike_addr.iike_ip.iike_ext_addr.iike_prefix_len contains a single byte prefix length. The prefix length must be from 1 to 128.</li></ul>

**Table 8-2. IKE\_POLICY (cont.)**

Member	Description
ike_my_id	<p>(IKE_IDENTIFIER) The identification of the local node used during the initial Exchange. This data structure has the same members as <a href="#">IKE_POLICY_SELECTOR/IKE_IDENTIFIER</a>. Only the following address types are allowed in this data member:</p> <ul style="list-style-type: none"> <li>• Single IPv4 address (IKE_IPV4 for IKEv1 or IKE2_ID_TYPE_IPV4_ADDR for IKEv2). ike_addr.ike_ip.ike_addr1 contains the IP address.</li> <li>• Single IPv6 address (IKE_IPV6 for IKEv1 or IKE2_ID_TYPE_IPV6_ADDR for IKEv2). ike_addr.ike_ip.ike_addr1 contains the IP address.</li> <li>• Domain Name (IKE_DOMAIN_NAME for IKEv1 or IKE2_ID_TYPE_FQDN for IKEv2). ike_addr.ike_domain points to the domain name string. Only supported in IKEv2 or IKEv1 Aggressive Mode. This must not be used if Main Mode exchange is allowed in ike_phase1_xchg.</li> <li>• User Domain Name (IKE_USER_DOMAIN_NAME for IKEv1 or IKE2_ID_TYPE_RFC822_ADDR for IKEv2). ike_addr.ike_domain points to the user domain name string in the format of an e-mail address. Only supported in IKEv2 or in IKEv1 Aggressive Mode. This must not be used if Main Mode exchange is allowed in ike_phase1_xchg.</li> <li>• Certificate "Distinguished Name" (DN) (IKE2_ID_TYPE_DER_ASN1_DN for IKEv2). This is only supported for IKEv2 policies. The ike_addr.ike_domain member points to the name.</li> </ul>

Table 8-2. IKE\_POLICY (cont.)

Member	Description
ike_peers_id	<p>(IKE_IDENTIFIER) The identification of the remote node used for identity verification during the Phase 1 Exchange. This data structure has the same members as IKE_POLICY_SELECTOR, defined below. Only the following address types are allowed in this data member:</p> <ul style="list-style-type: none"><li>• Single IPv4 address (IKE_IPV4 for IKEv1 or IKEv2_ID_TYPE_IPV4_ADDR for IKEv2). ike_addr.iike_ip.iike_addr1 contains the IP address.</li><li>• Range of IPv4 addresses (IKE_IPV4_RANGE for IKEv1). IP addresses between ike_addr.iike_ip.iike_addr1 and ike_addr.iike_ip.iike_ext_addr.iike_addr2 inclusive.</li><li>• IPv4 subnet (IKE_IPV4_SUBNET for IKEv1). ike_addr.iike_ip.iike_addr1 contains the subnet address and ike_addr.iike_ip.iike_ext_addr.iike_addr2 contains the subnet mask.</li><li>• Single IPv6 address (IKE_IPV6 for IKEv1 or IKEv2_ID_TYPE_IPV6_ADDR for IKEv2). ike_addr.iike_ip.iike_addr1 contains the IP address.</li><li>• Range of IPv6 addresses (IKE_IPV6_RANGE for IKEv1). IP addresses between ike_addr.iike_ip.iike_addr1 and ike_addr.iike_ip.iike_ext_addr.iike_addr2 inclusive.</li><li>• IPv6 subnet (IKE_IPV6_SUBNET for IKEv1). ike_addr.iike_ip.iike_addr1 contains the subnet address and ike_addr.iike_ip.iike_ext_addr.iike_prefix_len contains a single byte prefix length. The prefix length must be from 0 to 128.</li><li>• Domain Name (IKE_DOMAIN_NAME for IKEv1 or IKE2_ID_TYPE_FQDN for IKEv2). ike_addr.iike_domain points to the domain name string.</li><li>• User Domain Name (IKE_USER_DOMAIN_NAME for IKEv1 or IKE2_ID_TYPE_RFC822_ADDR for IKEv2). ike_addr.iike_domain points to the user domain name string in the format of an e-mail address.</li></ul> <p>Certificate "Distinguished Name" (DN) (IKE2_ID_TYPE_DER_ASN1_DN for IKEv2). This is only supported for IKEv2 policies. The ike_addr.iike_domain member points to the name.</p>
ike_ids	<p>(IKE_IPS_ID *) An array that specifies the ports and protocols for which Phase 2 Security Associations can be negotiated by this policy. This array must contain at least one element for the default identification.</p>

**Table 8-2. IKE\_POLICY (cont.)**

Member	Description
ike_xchg1_attris	<p>(<a href="#">IKE_ATTRIB</a> *) The Phase 1 Exchange Attributes. An array of a set of attributes that can be negotiated. This array is sorted in descending order of preference (i.e. the first element in the array provides a set of attributes that are most preferred and so on).</p> <p>If <code>ike_phase1_xchg</code> allows Aggressive Mode, then all items in this array must specify the same Oakley Group, since multiple choices of this parameter are not permissible in Aggressive Mode.</p>
ike_phase1_xchg	<p>This data member is IKEv1 specific. A flag that indicates the permissible Phase 1 Exchange Modes. The following are the valid flags in descending order of preference (For example, if multiple flags are set, IKE prefers the modes that appear first in the list. This order can be modified by the <code>IKE_PHASE1_DEFAULT_XCHG</code> configuration macro.). Multiple flags can be set by bit-wise OR'ing them together. At least one of Main Mode or Aggressive Mode must be enabled. Informational Mode is optional.</p> <ul style="list-style-type: none"> <li>• Main Mode (<code>IKE_XCHG_MAIN_FLAG</code>)</li> <li>• Aggressive Mode (<code>IKE_XCHG_AGGR_FLAG</code>)</li> <li>• Informational (<code>IKE_XCHG_INFO_FLAG</code>)</li> </ul>
ike_ids_no	The number of elements in <code>ike_ids</code> . This must be at least 1.
ike_xchg1_attris_no	The number of elements in <code>ike_xchg1_attris</code> . This must be at least 1 and the maximum limit is defined by the <code>IKE_MAX_TRANSFORMS</code> configuration macro.



**Table 8-2. IKE\_POLICY (cont.)**

Member	Description
ike_flags	<p>Flags specifying additional options for this policy. The following flag may be set:</p> <ul style="list-style-type: none"> <li>• Verify peer's identification data in Phase 1 (IKE_VERIFY_ID). If not set, the exchange proceeds even if the identity of the remote node does not match the expected identity specified in ike_peers_id.</li> <li>• IKE_VERIFY_CERT (only used when authenticating with RSA Signatures). When set, information from the Certificate is matched against the contents of the Identification payload.</li> <li>• IKE_VERIFY_AGAINST_CRL (only used when authenticating with RSA Signatures). When set, the peer's Certificate is checked against the certificate revocation list (CRL).</li> <li>• IKE_CA_IN_CERTREQ (only used when authenticating with RSA Signatures). When set, the Certification Authority's Distinguished Name from the certificate is sent to the peer in the Certificate Request.</li> <li>• IKE_INBAND_CERT_XCHG (only used when authenticating with RSA Signatures). When set, the certificate is exchanged in-band in IKE exchange. If not set, the certificate request is not sent and the peer's certificate is used from a locally specified file.</li> <li>• IKE_SEND_CERT_PROACTIVELY (only used when authenticating with RSA Signatures). When set, the Certificate is sent regardless of the fact whether the Certificate request is received or not.</li> </ul>
ike_version (Only available when IKEv2 is enabled)	<p>Specifies the major version of IKE to be used. This must be set to IKE_VERSION_2 to use IKEv2 with a policy. If left un-initialized, it defaults to IKE_VERSION_1.</p>
ike2_flags (Only available when IKEv2 is enabled)	<p>Policy flags specific to IKEv2. The following are possible values for this field:</p> <p>IKE2_SEND_TS_PAIR (Send a pair of IP addresses as TS. This cannot be specified along with the TS_RANGE flag)</p> <p>IKE2_SEND_TS_RANGE (Send a range of IP addresses. Currently the subnet to which the current address belongs is sent. Cannot be specified with TS_PAIR flag.</p> <p>IKE2_SA_DELETE (On expiration of an IKE SA, delete it and any child SAs established under it).</p> <p>IKE2_SA_REKEY (On expiration of IKE SA, re-key it with peer. Child SAs are inherited to the new IKE SA and do not need to be re-keyed unless they themselves expire. Cannot be specified along with IKE2_SA_DELETE.)</p>

**Table 8-2. IKE\_POLICY (cont.)**

Member	Description
ike2_sa_timeout (Only available when IKEv2 is enabled)	Specifies the timeout value for IKE SA. For IKEv2, timeouts are not negotiated and are statically configured on each end node. After this much time has elapsed, the SA is either deleted or re-keyed as the policy dictates.

## Related Topics

[IKE Data Structures](#)

[IKE\\_Add\\_Policy](#)

## IKE\_POLICY\_SELECTOR/IKE\_IDENTIFIER

IKE\_POLICY\_SELECTOR is the typedef for the ike\_policy\_selector data structure. The structure definition of IKE\_POLICY\_SELECTOR and IKE\_IDENTIFIER is the same, although their purpose is different. IKE\_POLICY\_SELECTOR is used to match a policy with a request to negotiate the IKE Security Associations. IKE\_IDENTIFIER is used in the IKE policy to specify identification data of local and remote hosts.

```
typedef struct ike_policy_selector
{
    union
    {
        IKE_SELECTOR_IP    ike_ip;
        CHAR                *ike_domain;
    } ike_addr;

    UINT8 ike_type;
} IKE_POLICY_SELECTOR;
```

The members of this structure are described in [Table 8-3](#) . Some of the members may have been omitted, as they are used internally.

**Table 8-3. IKE\_POLICY\_SELECTOR/IKE\_IDENTIFIER**

Member	Description
ike_addr	<p>A union which is defined as follows:</p> <pre>union {     <a href="#">IKE_SELECTOR_IP</a> ike_ip;     CHAR                *ike_domain;  } ike_addr;</pre> <p>ike_ip is used when the selector is a single IP address, a range of IP addresses or a subnet and ike_domain is used when the selector type is a Domain Name.</p>

**Table 8-3. IKE\_POLICY\_SELECTOR/IKE\_IDENTIFIER (cont.)**

Member	Description
ike_type	<p>A flag indicating the type of IP address being specified. The following are the valid values (only one of these should be set):</p> <ul style="list-style-type: none"><li>• Any IP address (IKE_WILDCARD).</li><li>• Single IPv4 address (IKE_IPV4/IKE2_ID_TYPE_IPV4_ADDR). ike_addr.ike_ip.ike_addr1 contains the IP address.</li><li>• Range of IPv4 addresses (IKE_IPV4_RANGE). IP addresses between ike_addr.ike_ip.ike_addr1 and ike_addr.ike_ip.ike_ext_addr.ike_addr2 inclusive.</li><li>• IPv4 subnet (IKE_IPV4_SUBNET). ike_addr.ike_ip.ike_addr1 contains the subnet address and ike_addr.ike_ip.ike_ext_addr.ike_addr2 contains the subnet mask.</li><li>• Single IPv6 address (IKE_IPV6/IKE2_ID_TYPE_IPV6_ADDR). ike_addr.ike_ip.ike_addr1 contains the IP address.</li><li>• Range of IPv6 addresses (IKE_IPV6_RANGE). IP addresses between ike_addr.ike_ip.ike_addr1 and ike_addr.ike_ip.ike_ext_addr.ike_addr2 inclusive.</li><li>• IPv6 subnet (IKE_IPV6_SUBNET). ike_addr.ike_ip.ike_addr1 contains the subnet address and ike_addr.ike_ip.ike_ext_addr.ike_prefix_len contains a single byte prefix length. The prefix length must be from 1 to 128.</li><li>• Domain Name (IKE_DOMAIN_NAME/IKE2_ID_TYPE_FQDN). ike_addr.ike_domain points to the domain name string. This address type is only used in the IKE_IDENTIFIER structure.</li><li>• User Domain Name (IKE_USER_DOMAIN_NAME/IKE2_ID_TYPE_RFC_822_ADDR). ike_addr.ike_domain points to the user domain name string, in the format of an e-mail address. This address type is only used in the IKE_IDENTIFIER structure.</li><li>• Certificate "Distinguished Name" (DN) (IKE2_ID_TYPE_DER_ASN1_DN for IKEv2). This is only supported by IKEv2.</li></ul>

## Related Topics

[IKE Data Structures](#)

[IKE\\_Get\\_Policy\\_Index](#)

[IKE\\_Get\\_Preshared\\_Key\\_Index](#)

## IKE\_SELECTOR\_IP

IKE\_SELECTOR\_IP is the typedef for the ike\_selector\_ip data structure. The structure is used for specifying an IP address type in the selector. Following is the definition of this data structure:

```
typedef struct ike_selector_ip
{
    UINT8          ike_addr1[MAX_ADDRESS_SIZE];
    union
    {
        UINT8      ike_addr2[MAX_ADDRESS_SIZE];
        UINT8      ike_prefix_len;
    } ike_ext_addr;
} IKE_SELECTOR_IP;
```

The members of this structure are described in :[Table 8-4](#)

**Table 8-4. IKE\_SELECTOR\_IP**

Member	Description
ike_addr1	IP address. This specifies either a single IP address, the first address of an IP range or a subnet address.
ike_ext_addr	<p>A union which is defined as follows:</p> <pre>union {     UINT8 ike_addr2[MAX_ADDRESS_SIZE];     UINT8 ike_prefix_len; } ike_ext_addr;</pre> <p>ike_addr2 specifies the last address of an IP range or to an IPv4 subnet mask. ike_prefix_len specifies an IPv6 prefix length when using IPv6 subnets. The prefix length must be from 1 to 128.</p>

## Related Topics

[IKE Data Structures](#)[IKE\\_POLICY\\_SELECTOR/IKE\\_IDENTIFIER](#)

## IKE\_IPS\_ID

IKE\_IPS\_ID is the typedef for the `ike_ips_id` data structure. This structure is used to define additional restrictions on the port or higher-layer protocol for which a policy is permitted to negotiate Phase 2 Security Associations.

```
typedef struct ike_ips_id
{
    UINT16      ike_port;
    UINT8       ike_protocol_id;
} IKE_IPS_ID;
```

The members of this structure are described in [Table 8-5](#):

**Table 8-5. IKE\_IPS\_ID**

Member	Description
ike_port	TCP or UDP port number. IKE allows negotiation of IPsec SAs that protect IP traffic destined to this port. IKE_WILDCARD can be used to allow negotiation of IPsec SAs for all ports.
ike_protocol_id	Higher layer protocol identifier. IKE allows negotiation of IPsec SAs that protect IP traffic of this protocol. The following are the valid values: <ul style="list-style-type: none"><li>• TCP protocol (IP_TCP_PROT)</li><li>• UDP protocol (IP_UDP_PROT)</li><li>• ICMP protocol (IP_ICMP_PROT)</li><li>• ICMP protocol (IP_ICMPV6_PROT)</li><li>• Match all protocols (IKE_WILDCARD)</li></ul>

## Related Topics

[IKE Data Structures](#)[IKE\\_POLICY](#)

## IKE\_ATTRIB

IKE\_ATTRIB is the typedef for the `ike_attrib` data structure. This structure is used to define the attributes for a Phase 1 Exchange. The following is a description of this structure:

```
typedef struct ike_attrib
{
    IKE_SA_LIFETIME ike_sa_lifetime;
    #if (IKE_INCLUDE_SIG_AUTH == NU_TRUE)
        UINT8      *ike_local_cert_file;
```

```

        UINT8      *ike_local_key_file;
        UINT8      *ike_ca_cert_file;
        UINT8      *ike_peer_cert_file;
        UINT8      *ike_crl_file;
    #if (IKE_INCLUDE_PEM == NU_TRUE)
        pem_password_cb *ike_pem_callback;
    #endif /* #if (IKE_INCLUDE_PEM == NU_TRUE) */
    #endif /* #if (IKE_INCLUDE_SIG_AUTH == NU_TRUE) */
        UINT8      *ike_remote_key;
        UINT16     ike_remote_key_len;
        UINT16     ike_encryption_algo;
        UINT16     ike_auth_method;
        UINT16     ike_hash_algo;
        UINT16     ike_key_len;
        UINT16     ike_group_desc;
        UINT8      ike_cert_encoding;
    #if (IKE_INCLUDE_VERSION_2 == NU_TRUE)
        UINT8      ike_pad[1];
        UINT16     ike2_prf_algo;
        UINT8      *ike2_psk_local;
        UINT8      *ike2_psk_remote;
        UINT16     ike2_psk_local_len;
        UINT16     ike2_psk_remote_len;
    #else
        UINT8      ike_pad[3];
    #endif
} IKE_ATTRIB;

```

The members of this structure are described in [Table 8-6](#). Some of the members were omitted as they are used internally:

**Table 8-6. IKE\_ATTRIB**

Member	Description
ike_sa_lifetime	( <a href="#">IKE_SA_LIFETIME</a> ) Minimum lifetime for the IKE Security Association that can be negotiated.
ike_local_cert_file	Name of the certificate file used when authenticating with RSA Signatures.
ike_local_key_file	Name of private key file to use when authenticating with RSA Signatures.
ike_ca_cert_file	Name of the CA's certificate file (when authenticating with RSA Signature). Note that if <code>IKE_INCLUDE_SIG_AUTH</code> is enabled in the configuration and if the "ike_auth_method" member is set to something other than RSA, this data member must still be specified and must point to a valid CA certificate file. This case is because the remote side may choose to use RSA regardless of the local node's configuration.
ike_peer_cert_file	Name of the peer's certificate file. This is used when authentication is to be done by RSA Signature, but in-band certificate exchange is disabled.

Table 8-6. IKE\_ATTRIB (cont.)

Member	Description
ike_crl_file	CRL file to check peer's certificate against when authenticating with RSA Signature and certificate verification is enabled. (Certificate verification can be enabled by setting the appropriate flag in the <a href="#">IKE_POLICY</a> structure.)
pem_callback	<p>Base64 encoded and encrypted private key file can only be read by supplying the password to decrypt the contents. This function pointer to the callback function reads the password to read the Base64 encrypted private key file. This field is available only when support for PEM format is enabled. Signature of this function is as follows:</p> <pre>int pem_password_cb(char *buf, int size, int rwflag, void *userdata)</pre> <ul style="list-style-type: none"><li>• buf: Pre-allocated buffer to hold the password on return. The size of this buffer is given in the size parameter.</li><li>• size: The size of the buffer. The callback function is called with a value of 1024. Passwords longer than this are not supported.</li><li>• rwflag: Not used by IKE.</li></ul> <p>userdata: Not used by IKE.</p>
ike_remote_key	Remote host's public key for the Signature algorithm, in ASN1 format. This is required only if the authentication method specified in ike_auth_method is based on Signatures.
ike_remote_key_len	Length of ike_remote_key, in bytes.
ike_encryption_algo	<p>The encryption algorithm to be used. The following are valid values:</p> <ul style="list-style-type: none"><li>• DES-CBC (IKE_DES)</li><li>• 3DES-CBC (IKE_3DES)</li><li>• Blowfish-CBC (IKE_BLOWFISH)</li><li>• CAST-CBC (IKE_CAST_128)</li><li>• AES-CBC (IKE_AES)</li></ul> <p>When using IKEv2, the following are valid values:</p> <ul style="list-style-type: none"><li>• IKE2_ENCR_DES</li><li>• IKE2_ENCR_3DES</li><li>• IKE2_ENCR_CAST</li><li>• IKE2_ENCR_BLOWFISH</li><li>• IKE2_ENCR_AES_CBC</li></ul>

**Table 8-6. IKE\_ATTRIB (cont.)**

Member	Description
ike_auth_method	<p>Authentication method to be used in Phase 1. The following are valid values:</p> <ul style="list-style-type: none"> <li>• Authentication using Pre-shared Keys (IKE_PSK). At least one valid pre-shared key must be added to the database.</li> <li>• Authentication using RSA Signatures (IKE_RSA). Local and Certification Authority's certificates should be specified.</li> </ul> <p>When using IKEv2, the following values should be used instead:</p> <ul style="list-style-type: none"> <li>• IKE2_AUTH_METHOD_RSA_DS</li> <li>• IKE2_AUTH_METHOD_SKEY_MIC</li> </ul>
ike_hash_algo	<p>The hash algorithm to be used. The following are valid values:</p> <ul style="list-style-type: none"> <li>• MD5 (IKE_MD5)</li> <li>• SHA1 (IKE_SHA1)</li> </ul> <p>When IKEv2 is used, the following are the valid values:</p> <ul style="list-style-type: none"> <li>• IKE2_AUTH_HMAC_MD5_96</li> <li>• IKE2_AUTH_HMAC_SHA1_96</li> <li>• IKE2_AUTH_AES_XCBC_96</li> </ul>
ike_key_len	<p>The minimum permissible encryption algorithm key length that can be negotiated for the IKE Security Association, specified in bytes. Setting this to IKE_WILDCARD uses the default key length for each algorithm. The following are valid values:</p> <ul style="list-style-type: none"> <li>• 8 bytes for DES-CBC (default 8)</li> <li>• 24 bytes for 3DES-CBC (default 24)</li> <li>• 5 to 56 bytes for Blowfish-CBC (default 16)</li> <li>• 5 to 16 bytes for CAST-CBC (default 16)</li> <li>• 16, 24 or 32 bytes for AES-CBC (default 16)</li> </ul>
ike_group_desc	<p>Oakley Group used in the Diffie-Hellman key exchange during the initial IKEv1 or IKEv2 exchange. The following are valid values:</p> <ul style="list-style-type: none"> <li>• IKE_GROUP_MODP_768</li> <li>• IKE_GROUP_MODP_1024</li> <li>• IKE_GROUP_MODP_1536</li> <li>• IKE_GROUP_MODP_2048</li> <li>• IKE_GROUP_MODP_3072</li> <li>• IKE_GROUP_MODP_4096</li> <li>• IKE_GROUP_MODP_6144</li> <li>• IKE_GROUP_MODP_8192</li> <li>• IKE_GROUP_MODP_1024_160_POS (for IKEv2 only)</li> <li>• IKE_GROUP_MODP_2048_224_POS (for IKEv2 only)</li> <li>• IKE_GROUP_MODP_2048_256_POS (for IKEv2 only)</li> </ul>



**Table 8-6. IKE\_ATTRIB (cont.)**

Member	Description
ike_cert_encoding	Encoding of the certificate files to be used. All the certificate and key files should have the same encoding. Its values can be either <code>IKE_X509_FILETYPE_ASN1</code> or <code>IKE_X509_FILETYPE_PEM</code> . Note that if <code>IKE_INCLUDE_SIG_AUTH</code> is enabled in the configuration and if the "ike_auth_method" member is set to something other than RSA, this data member must still be correctly specified. This case is because the remote side may choose to use RSA regardless of the local node's configuration.
ike2_prf_algo	Specifies the PRF algorithm to be used. Valid values are: <ul style="list-style-type: none"><li>• <code>IKE2_PRF_HMAC_MD5</code></li><li>• <code>IKE2_PRF_HMAC_SHA1</code></li><li>• <code>IKE2_PRF_AES128_XCBC</code></li></ul>
ike2_psk_local	Pre-shared key for the local node
ike2_psk_remote	Pre-shared key for the remote node
ike2_psk_local_len	Length of the local node's Pre-shared key
ike2_psk_remote_len	Length of the remote node's Pre-shared key

## Related Topics

[IKE Data Structures](#)[IKE\\_POLICY](#)

## IKE\_SA\_LIFETIME

`IKE_SA_LIFETIME` is the typedef for the `ike_sa_lifetime` data structure. This structure is used for specifying the lifetime of an IKE SA. The following is the description for this structure:

```
typedef struct ike_sa_lifetime
{
    UINT32    ike_no_of_secs;
} IKE_SA_LIFETIME;
```

The members of this structure are described in [Table 8-7](#):

**Table 8-7. IKE\_SA\_LIFETIME**

Member	Description
ike_no_of_secs	(UINT32) Lifetime in seconds. Valid range is from 1 to 4,294,967,295. Setting this to IKE_WILDCARD accepts any lifetime when responding to an exchange and assumes a default value of 28800 seconds (8 hours) when initiating an exchange.

## Related Topics

[IKE Data Structures](#)

## IKE\_KEY\_PAIR

IKE\_KEY\_PAIR is the typedef for the ike\_key\_pair data structure. This structure is used for specifying public and private Signature keys. The following is the definition for this structure:

```
typedef struct ike_key_pair
{
    UINT8      *ike_public_key;
    UINT8      *ike_private_key;
    UINT16     ike_public_key_len;
    UINT16     ike_private_key_len;
} IKE_KEY_PAIR;
```

The members of this structure are described in [Table 8-8](#):

**Table 8-8. IKE\_KEY\_PAIR**

Member	Description
ike_public_key	Local public key of the Signature algorithm, in ASN1 format.
ike_private_key	Local private key of the Signature algorithm, in ASN1 format.
ike_public_key_len	Length of the public key, in bytes.
ike_private_key_len	Length of the private key, in bytes.

## Related Topics

[IKE Data Structures](#)

## IKE\_PRESHARED\_KEY

IKE\_PRESHARED\_KEY is the typedef for the ike\_preshared\_key data structure. This structure is used to define a pre-shared key.

```
typedef struct ike_preshared_key
{
    IKE_IDENTIFIER    ike_id;
    UINT8             *ike_key;
    UINT8             ike_key_len;
} IKE_PRESHARED_KEY;
```

The members of this structure are described in [Table 8-9](#). Some of the members may have been omitted, as they are used internally by IKE:

**Table 8-9. IKE\_PRESHARED\_KEY**

Member	Description
ike_id	Identifier of this pre-shared key. The description of this structure is similar to the <a href="#">IKE_POLICY_SELECTOR/IKE_IDENTIFIER</a> defined in the "IKE Policy" section of this chapter.
ike_key	Key material for this pre-shared key. There are no restrictions on the format of the key material.
ike_key_len	Length of the key material specified in ike_key, in bytes. Valid range is from 1 to 255.

## Related Topics

[IKE Data Structures](#)

[IKE\\_Add\\_Preshared\\_Key](#)

## IKE\_INITIATE\_REQ

The IKE\_INITIATE\_REQ is the typedef for the ike\_initiate\_req data structure. This structure is used to specify a request for an IKE exchange initiation.

```
typedef struct ike_initiate_req
{
    IPSEC_SECURITY_PROTOCOL ike_ips_security;
    IPSEC_SELECTOR          ike_ips_select;
    UNSIGNED                ike_suspend;
    UINT32                  ike_dev_index;
} IKE_INITIATE_REQ;
```

The members of this structure are described in [Table 8-10](#) :

**Table 8-10. IKE\_INITIATE\_REQ**

Member	Description
ike_ips_security	An IPsec security protocol to be negotiated by IKE. This must be exactly the same as one of the security protocols specified in the Nucleus IPsec policy. If the requested security protocol is part of a larger protocol suite, then all required protocols are negotiated.
ike_ips_select	Nucleus IPsec Selector of the SAs that are established as a result of the IKE exchange. Refer to the <a href="#">IPsec</a> chapter for a description of this structure. This selector must always specify single IP addresses for both source and destination, because IPsec SA selectors must always be based on single IP addresses. The only exception is when requesting an IKE exchange for a tunnel that carries traffic for an IP range or an IP subnet.
ike_suspend	Specifies whether the caller should be suspended for as long as the IKE exchange continues or whether the exchange should continue in the background after control to the caller is returned. The following are valid values: <ul style="list-style-type: none"><li>• Blocking request (NU_SUSPEND)</li><li>• Non-blocking request (NU_NO_SUSPEND)</li></ul> Number of seconds. Request blocking for a specified number of seconds and if the exchange does not complete within this time, then let it continue in the background.
ike_dev_index	Index of the network interface used to perform the IKE exchange. This interface must be associated with both an IKE and an IPsec group.

## Related Topics

[IKE Data Structures](#)

[IKE\\_Initiate](#)

## Pre-shared Keys

### Note



This section is only applicable to IKEv1. Note that IKEv2 does not have an independent pre-shared keys database. IKEv2 stores the pre-shared keys of the local and remote nodes within the Policy structure. For more information please see the description of "*ike2\_psk\_local*" and "*ike2\_psk\_remote*" fields in the [IKE\\_ATTRIB](#) structure.

---

Authentication based on pre-shared keys involves sharing of a common secret between the two peers performing an IKE exchange. Both peers identify themselves during the exchange. When identification data is received, the pre-shared keys database is searched for a matching key.

[Table 8-11](#) shows possible methods of identification:

**Table 8-11. Identification Methods**

Identification Type	Description
IP Address	Each host uses its IP address to identify itself to the remote host. This method is most commonly used and is the only choice when using Main Mode exchange.
Domain names (FQDN)	A fully qualified domain name is used for identification. This method is allowed only in Aggressive Mode.
User domain names (user FQDN)	The user domain name in the format of an e-mail address is used for identification. This is useful if multiple users have different pre-shared keys on a single host. This method is allowed only in Aggressive Mode.

In IKE, the previously mentioned identification data for an exchange is specified in the IKE policy. The `ike_my_id` data member of the policy structure contains the identification to be sent to the remote host, and the `ike_peers_id` data member contains an identifier used to verify the identity of the remote host before it is used to look up a pre-shared key. IKE maintains a list of pre-shared keys. Each pre-shared key is composed of two data members. First, the key material that has been shared between the local and remote host, and second, an identifier which specifies one or more remote hosts to which the key applies. The identifier could be a single IP address, a range of IP addresses, a subnet, a domain name, a user domain name, or a wildcard that matches all addresses. All pre-shared keys in the database are stored in the same order as the one in which they are added to the database. New keys are appended to the end of the pre-shared key list.

Host identities are searched in the pre-shared keys list starting from the first item. [Table 8-12](#) gives an example of a pre-shared keys list:

**Table 8-12. Pre-shared Keys**

Identifier Type	Value
Single IP Address	192.168.0.1
User domain name	"first_user@mentor.com"
Range of IP Addresses	192.168.0.20 to 192.168.0.30
Range of IP Addresses	192.168.0.1 to 192.168.0.99
Subnet	192.168.0.0 Subnet Mask 255.255.255.0
Single IP Address	192.168.0.2
User domain name	"second_user@mentor.com"


**Table 8-12. Pre-shared Keys (cont.)**

Identifier Type	Value
Wildcard	Value not applicable. Matches all addresses.

The list is searched from the beginning. If multiple pre-shared keys match the selection criteria, the key that occurs the earliest in the list is selected. In the previous example, an IKE node with an IP address identity 192.168.0.1 matches the first, fourth, fifth, and eighth keys. However, the first key is used during negotiation since it is the first match.

---


**Note**

 The Wildcard policy matches all addresses so it is usually the last policy to be added to the list. Any policy following it will never be matched.

---

---

**Note**

 The [IKE\\_PRESHARED\\_KEY](#) data structure is used in IKE API functions related to pre-shared keys. This structure is located in the [IKE Data Structures](#) section.


---

## IKE API Functions

This section discusses the APIs provided for the creation of IKE policies and SAs. The [IKE API Usage Example](#) section creates an IKE environment using the services listed here.

---


**Note**

 To make use of the APIs, you must include the *os/include/networking/nu\_networking.h* header file.

---

---

**Note**

 All IKE API functions must be called from a task thread, because IKE uses blocking semaphores to protect its critical regions.

---

- [IKE\\_Add\\_Group](#)
- [IKE\\_Add\\_Policy](#)
- [IKE\\_Add\\_Preshared\\_Key](#)
- [IKE\\_Add\\_To\\_Group](#)
- [IKE\\_Get\\_Group](#)
- [IKE\\_Get\\_Group\\_Opt](#)
- [IKE\\_Get\\_Policy\\_Index](#)
- [IKE\\_Get\\_Policy\\_Opt](#)

- [IKE\\_Get\\_Preshared\\_Key\\_Index](#)
- [IKE\\_Initiate](#)
- [IKE\\_Remove\\_From\\_Group](#)
- [IKE\\_Remove\\_Group](#)
- [IKE\\_Remove\\_Policy](#)
- [IKE\\_Remove\\_Preshared\\_Key](#)
- [IKE\\_Set\\_Policy\\_Opt](#)
- [IKE\\_Shutdown](#)

## IKE\_Add\_Group

This function adds a new IKE group to the database. The group is initially empty and contains no interfaces or policies.

### Usage

```
STATUS IKE_Add_Group (CHAR *group_name);
```

### Arguments

- `group_name`  
Pointer to the name of the group to be added. This must be a maximum of `IKE_MAX_GROUP_NAME_LEN` characters, defined in the IKE configuration. This limit includes the null-terminator.

### Return Values

- `NU_SUCCESS`  
The group was successfully added.
- `NU_TIMEOUT`  
Timed out waiting for a resource.
- `NU_NO_MEMORY`  
Not enough memory to carry out the request.
- `IKE_INVALID_PARAMS`  
The input parameter is invalid.
- `IKE_ALREADY_EXISTS`  
A group with the specified name already exists.
- `IKE_LENGTH_IS_LONG`  
Length of the group name is too long.

### Example

```
/* Local variable to store the return status. */
STATUS status;

/* Add a new group to the database. */
status = IKE_Add_Group("ike_test_group");

if(status == NU_SUCCESS)
{
    /* Group added successfully. */
}
```



## Related Topics

[IKE API Functions](#)

## IKE\_Add\_Policy

This function adds a new policy for IKE.

### Usage

```
STATUS IKE_Add_Policy (CHAR      *group_name,  
                      IKE_POLICY *policy,  
                      UINT32     *index);
```

### Arguments

- **group\_name**  
Pointer to the IKE group name to which the new policy is added.
- **policy**  
A pointer to the [IKE\\_POLICY](#) structure that contains the values to copy to the new policy. The [IKE\\_POLICY](#) data structure is described in the [IKE Data Structures](#) section.
- **index**  
When the function returns, this variable contains the unique identifier assigned to the new policy.

### Return Values

- **NU\_SUCCESS**  
Policy was successfully added.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **NU\_NO\_MEMORY**  
Indicates that there is not enough memory.
- **IKE\_INVALID\_KEYLEN**  
Key length of the encryption algorithm is either invalid or not supported.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_ALREADY\_EXISTS**  
Indicates that a policy with the same selector already exists.
- **IKE\_NOT\_FOUND**  
The IKE group was not found.
- **IKE\_UNSUPPORTED\_ALGO**  
An algorithm specified in the policy is not supported.

## Description

Memory is allocated for the policy structure within this function, and the values are copied from the passed structure. Therefore, the passed structure can have local scope. See “[IKE Policy](#)” on page 1410 for more information on IKE policies. Also, refer to “[IKE Policy](#)” on page 1340 for a more in-depth explanation of how IKE proposals are derived from policies.

This function returns a locally assigned unique identifier for the policy.

## Example

```
/* Local variable to store the return status */
STATUS      status;

/* Local variable to store the new policy */
IKE_POLICY   policy;

/* Phase 2 ID allowed by the policy */
IKE_IPS_ID   phase2_id;

/* Alternative Phase 1 attributes to be negotiated in phase 1 */
IKE_ATTRIB   attribs[2];

/* Unique identifier of the policy */
UINT32       policy_index = 0;

/* Assuming that the local IP address is 192.168.0.1 */
UINT8        local_ip_addr[] = { 192, 168, 0, 1 };

/* Assuming that the remote IP address is 192.168.0.20 */
UINT8        foreign_ip_addr[] = { 192, 168, 0, 20 };

/* Set the selector so that it selects ALL addresses. */
policy.ike_select.ike_type = IKE_WILDCARD;

/* Set the local Identifier. */
policy.ike_my_id.ike_type = IKE_IPV4;

memcpy(policy.ike_my_id.ike_addr.ike_ip.ike_addr1,
       local_ip_addr, IP_ADDR_LEN);

/* Set the remote Identifier. */
policy.ike_peers_id.ike_type = IKE_IPV4;

memcpy(policy.ike_peers_id.ike_addr.ike_ip.ike_addr1,
       foreign_ip_addr, IP_ADDR_LEN);

/* Allow all possible modes in phase 1. */
policy.ike_phase1_xchg = (IKE_XCHG_AGGR_FLAG |
                         IKE_XCHG_MAIN_FLAG |
                         IKE_XCHG_INFO_FLAG);

/* Allow all possible modes in phase 2. */
policy.ike_phase2_xchg = (IKE_XCHG_QUICK_FLAG |
                         IKE_XCHG_INFO_FLAG);

/* Set policy flags to zero. */
```

```
policy.ike_flags = 0;

/* Initialize first set of Phase 1 exchange attributes. */
attribs[0].ike_auth_method      = IKE_PSK;
attribs[0].ike_encryption_algo  = IKE_DES;
attribs[0].ike_key_len          = IKE_WILDCARD;
attribs[0].ike_hash_algo        = IKE_MD5;
attribs[0].ike_group_desc       = IKE_GROUP_MODP_768;
attribs[0].ike_sa_lifetime.ike_no_of_secs =
                                28800;

/* Pre_Shared Key for the policy attrib */
attribs[0].ike2_psk_local = "IKETEST12345678!";
attribs[0].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[0].ike2_psk_remote = "IKETEST12345678!";
attribs[0].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Initialize second set of Phase 1 exchange attributes. */
attribs[1].ike_auth_method      = IKE_PSK;
attribs[1].ike_encryption_algo  = IKE_BLOWFISH;
attribs[1].ike_hash_algo        = IKE_SHA1;
attribs[1].ike_key_len          = 16;
attribs[1].ike_group_desc       = IKE_GROUP_MODP_768;

/* Pre_Shared Key for the policy attrib */
attribs[1].ike2_psk_local = "IKETEST12345678!";
attribs[1].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[1].ike2_psk_remote = "IKETEST12345678!";
attribs[1].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the above exchange attributes in the policy. */
policy.ike_xchg1_attribs_no = 2;
policy.ike_xchg1_attribs    = attribs;

/* Allow all phase 2 exchanges for all protocols. */
phase2_id.ike_protocol_id = IKE_WILDCARD;
phase2_id.ike_port         = IKE_WILDCARD;

/* Set phase 2 protocols/ports allowed by the IKE policy. This could
 * be an array, but only a single wildcard entry would be enough. */
policy.ike_ids      = &phase2_id;
policy.ike_ids_no    = 1;

/* Create this policy. Policy index is returned. */
status = IKE_Add_Policy("ike_test_group", &policy,
                        &policy_index);

if(status == NU_SUCCESS)
{
    /* Successfully added IKE policy to the group. */
}
```

## Example

```
/* Local variable to store the return status */
STATUS status;
/* Local variable to store the new policy */
IKE_POLICY policy;
```

```
/* ID allowed by the policy for IPsec SAs */
IKE_IPS_ID ips_id;
/* Alternative IKE SA attributes to be negotiated for IKE SA */
IKE_ATTRIB attribs[1];

/* Unique identifier of the policy */
/* Assuming that the local IP address is 192.168.0.1 */
UINT8 local_ip_addr[] = { 192, 168, 0, 1 };
policy.iike2_flags =IKE2_SEND_TS_PAIR;

/* Set version of IKE to be used to version 2. */
policy.iike_version = IKE_VERSION_2;

/* Initialize the set of IKE exchange attributes. */
attribs[0].iike_auth_method= IKE_PSK;
attribs[0].iike_encryption_algo= IKE2_ENCR_3DES;
attribs[0].iike_key_len= IKE_WILDCARD;
attribs[0].iike_hash_algo= IKE2_AUTH_HMAC_MD5_96;
attribs[0].iike2_prf_algo      = IKE2_PRF_HMAC_MD5;
attribs[0].iike_group_desc     = IKE_GROUP_MODP_1024;
attribs[0].iike_sa_lifetime.iike_no_of_secs = IKE_WILDCARD;

/* Pre_Shared Key for the policy attrib */
attribs[0].iike2_psk_local = "IKETEST12345678!";
attribs[0].iike2_psk_local_len = strlen("IKETEST12345678!");
attribs[0].iike2_psk_remote = "IKETEST12345678!";
attribs[0].iike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the exchange attribute in the policy. Only a single
 * set of attributes is used.
 */
policy.iike_xchgl_attribs      = attribs;
policy.iike_xchgl_attribs_no = 1; /* Assuming that the remote IP address is
192.168.0.20 */
UINT8 foreign_ip_addr[] = { 192, 168, 0, 20 };

/* Set the selector so that it matches all addresses. */
policy.iike_select.iike_type = IKE_WILDCARD;

/* Set local and remote identification to match all addresses. */
policy.iike_my_id.iike_type      = IKE_IPv4;
policy.iike_peers_id.iike_type   = IKE_IPv4;

/* Copy the local and remote IPv4 addresses into the policy structure. */
memcpy(policy.iike_my_id.iike_addr.iike_ip.iike_addr1, local_ip_addr,
IP_ADDR_LEN);
memcpy(policy.iike_peers_id.iike_addr.iike_ip.iike_addr1, foreign_ip_addr,
IP_ADDR_LEN);

/* Initialize policy flags. */
policy.iike_flags = 0;

/* Set IKEv2 specific policy flags. */
policy.iike2_flags = IKE2_SEND_TS_PAIR;

/* Set the version of IKE to be used to version 2. */
```

```
policy.ike_version = IKE_VERSION_2;

/* Initialize the set of IKE exchange attributes. */
attribs[0].ike_auth_method= IKE_PSK;
attribs[0].ike_encryption_algo= IKE2_ENCR_3DES;
attribs[0].ike_key_len= IKE_WILDCARD;attribs[0].ike_hash_algo=
IKE2_AUTH_HMAC_MD5_96;
attribs[0].ike2_prf_algo      = IKE2_PRF_HMAC_MD5;
attribs[0].ike_group_desc     = IKE_GROUP_MODP_1024;
attribs[0].ike_sa_lifetime.ike_no_of_secs = IKE_WILDCARD;

/* Pre_Shared Key for the policy attrib */
attribs[0].ike2_psk_local = "IKETEST12345678!";
attribs[0].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[0].ike2_psk_remote = "IKETEST12345678!";
attribs[0].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the exchange attribute in the policy. Only a single set of
   attributes is used. */
policy.ike_xchgl_attribs      = attribs;
policy.ike_xchgl_attribs_no = 1;
/* Allow all child exchanges for all protocols. */
ips_id.ike_protocol_id = IKE_WILDCARD;
/* Local variable to store the return status */
STATUS status;

/* Local variable to store the new policy. */
IKE_POLICY policy

/* ID allowed by the policy for IPsec SAs */
IKE_IPS_ID ips_id;

/* Alternative IKE SA attributes to be negotiated for IKE SA */
IKE_ATTRIB attribs[1];

/* Unique identifier of the policy */
UINT32 policy_index = 0;

/* Assuming that the local IP address is 192.168.0.1 */
UINT8 local_ip_addr[] = { 192, 168, 0, 1 };

/* Assuming that the remote IP address is 192.168.0.20 */
UINT8 foreign_ip_addr[] = { 192, 168, 0, 20 };

/* Set the selector so that it matches all addresses. */
policy.ike_select.ike_type = IKE_WILDCARD;

/* Set local and remote identification to match all addresses. */
policy.ike_my_id.ike_type      = IKE_IPv4;
policy.ike_peers_id.ike_type   = IKE_IPv4;

/* Copy the local and remote IPv4 addresses into the policy structure. */
memcpy(policy.ike_my_id.ike_addr.ike_ip.ike_addr1, local_ip_addr,
IP_ADDR_LEN);
memcpy(policy.ike_peers_id.ike_addr.ike_ip.ike_addr1, foreign_ip_addr,
IP_ADDR_LEN);

/* Initialize policy flags. */
```

```
policy.ike_flags = 0;

/* Set IKEv2 specific policy flags. */
policy.ike2_flags = IKE2_SEND_TS_PAIR;

/* Set version of IKE to be used to version 2. */
policy.ike_version = IKE_VERSION_2;

/* Initialize the set of IKE exchange attributes. */
attribs[0].ike_auth_method = IKE_PSK;
attribs[0].ike_encryption_algo = IKE2_ENCR_3DES;
attribs[0].ike_key_len = IKE_WILDCARD;
attribs[0].ike_hash_algo = IKE2_AUTH_HMAC_MD5_96;
attribs[0].ike2_prf_algo = IKE2_PRF_HMAC_MD5;
attribs[0].ike_group_desc = IKE_GROUP_MODP_1024;
attribs[0].ike_sa_lifetime.ike_no_of_secs = IKE_WILDCARD;

/* Pre_Shared Key for the policy attrib */
attribs[0].ike2_psk_local = "IKETEST12345678!";
attribs[0].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[0].ike2_psk_remote = "IKETEST12345678!";
attribs[0].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the exchange attribute in the policy. Only a single
 * set of attributes is used.
 */
policy.ike_xchgl_attribs = attribs;
policy.ike_xchgl_attribs_no = 1;

/* Allow all child exchanges for all protocols. */
ips_id.ike_protocol_id = IKE_WILDCARD;
ips_id.ike_port = IKE_WILDCARD;

/* Create this policy. Policy index is returned. */
status = IKE_Add_Policy("ike_test_group", &policy, &policy_index);

if(status != NU_SUCCESS)
{
    printf("Failed to add IKE policy to group.\r\n");
}
status = IKE_Add_Policy("ike_test_group", &policy, &policy_index);

if(status != NU_SUCCESS)
{
    printf("Failed to add IKE policy to group.\r\n");
}
```

## Related Topics

### [IKE API Functions](#)

## IKE\_Add\_Preshared\_Key

This function adds a new pre-shared key to the database.

---

### Note



This API is specific to IKEv1 and should not be used for IKEv2. The pre-shared keys for IKEv2 are specified within the policy structure.

---

## Usage

```
STATUS IKE_Add_Preshared_Key (IKE_PRESHARED_KEY *psk,  
                             UINT16              *index);
```

## Arguments

- **psk**  
Pointer to the [IKE\\_PRESHARED\\_KEY](#) structure that contains the values to be copied to the new pre-shared key. The [IKE\\_PRESHARED\\_KEY](#) is described in the [IKE Data Structures](#) section.
- **index**  
On return, this contains a unique index assigned to the new pre-shared key.

## Return Values

- **NU\_SUCCESS**  
The request was successful.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_ALREADY\_EXISTS**  
A pre-shared key having the specified identifier already exists in the database.

## Description

Memory is allocated for the pre-shared key within this function, and the values are copied from the passed structure. Therefore, the passed pre-shared key structure can have local scope. See [“IKE Pre-shared Keys”](#) on page 1409 for more information on pre-shared keys.

This function returns a locally assigned unique identifier for the pre-shared key.

---

### Note



Pre-shared key search is dependent on the internal order of the keys maintained by IKE. The order in which keys are added to the database is important and it affects search results. Keys added first are matched first.

---



## Example

```
/* Local variable to store the return status */
STATUS status;

/* Variable to store the unique identifier */
UINT16 psk_index;

/* Variable used to store the pre-shared key before it is
 * added to the database.
 */
IKE_PRESHARED_KEY psk;

/* Create a pre-shared key for all IP addresses in the
 * range 192.168.0.1 to 192.168.0.50.
 */
psk.iike_id.iike_type = IKE_IPV4_RANGE;

/* Specify the IP range. */
psk.iike_id.iike_addr.iike_ip.iike_addr1[0] = 192;
psk.iike_id.iike_addr.iike_ip.iike_addr1[1] = 168;
psk.iike_id.iike_addr.iike_ip.iike_addr1[2] = 0;
psk.iike_id.iike_addr.iike_ip.iike_addr1[3] = 1;

psk.iike_id.iike_addr.iike_ip.iike_addr2[0] = 192;
psk.iike_id.iike_addr.iike_ip.iike_addr2[1] = 168;
psk.iike_id.iike_addr.iike_ip.iike_addr2[2] = 0;
psk.iike_id.iike_addr.iike_ip.iike_addr2[3] = 50;

/* Set the key material. */
psk.iike_key = "topsecretkey";
psk.iike_key_len = 12;

/* Add this pre-shared key to the database. */
status = IKE_Preshared_Key(&psk, &psk_index);

if(status != NU_SUCCESS)
{
    /* Unable to add the pre-shared key to the database. */
}

/* Create a pre-shared key for a user domain name. */
psk.iike_id.iike_type = IKE_USER_DOMAIN_NAME;

/* Specify the user domain name (USER_FQDN). */
psk.iike_id.iike_addr.iike_domain = "user@mentor.com";

/* Set the key material. */
psk.iike_key = "anothersecret";
psk.iike_key_len = 13;

/* Add the second pre-shared key to the database. */
status = IKE_Preshared_Key(&psk, &psk_index);

if(status != NU_SUCCESS)
{
    /* Unable to add the pre-shared key to the database. */
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Add\_To\_Group

This function adds a network interface to the specified IKE group. If the interface is already associated with a group, then this request fails.

### Usage

```
STATUS IKE_Add_To_Group (CHAR *group_name,  
                        CHAR *interface_name);
```

### Arguments

- **group\_name**  
Pointer to the name of the group with which the interface is to be associated.
- **interface\_name**  
Pointer to the name of the network interface. This must be the same name specified when the interface is initialized using the `NU_Init_Devices` function.

### Return Values

- **NU\_SUCCESS**  
The interface was successfully associated with the IKE group.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One or more of the input parameters are invalid.
- **IKE\_ALREADY\_EXISTS**  
Specified network interface is already registered with a group.
- **IKE\_NOT\_FOUND**  
Specified IKE group was not found in the database, or the specified interface does not exist.

### Example

```
/* Local variable to store the return status. */  
STATUS status;  
  
/* Structure used to initialize an interface. */  
NU_DEVICE test_device;  
  
/* Set the interface name. */  
test_device.dv_name = "eth0";  
  
/* Setting other target-specific members of the  
 * device structure.  
 */  
. . .  
  
/* Initialize the interface. */
```

```
status = NU_Init_Devices(&test_device, 1);

/* Initialization of Nucleus IPsec and IKE. */
. . .

/* Add a new group to the database. */
status = IKE_Add_Group("ike_test_group");

if(status == NU_SUCCESS)
{
    /* Add above initialized interface to the group. */
    status = IKE_Add_To_Group("ike_test_group", "eth0");

    if(status == NU_SUCCESS)
    {
        /* Interface has been registered successfully. */
    }
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Get\_Group

This function returns the name of the group that contains the specified interface.

### Usage

```
STATUS IKE_Get_Group (CHAR    *interface_name,
                     CHAR    *return_group,
                     UINT32  *total_len);
```

### Arguments

- **interface\_name**  
Pointer to the name of the interface to be found.
- **return\_group**  
Pointer to a buffer for storing the name of the group.
- **total\_len**  
The maximum size string that can be stored in **return\_group** including the null-terminator. On a successful return, this contains the size of the name copied to **return\_group**.

### Return Values

- **NU\_SUCCESS**  
The group was successfully found.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_NOT\_FOUND**  
The group or the interface was not found.
- **IKE\_LENGTH\_IS\_SHORT**  
Indicates that the value to be returned required more memory than was passed. The number of bytes that were required to be written are returned in **total\_len**.

### Example

```
/* Local variable to store the return status */
STATUS status;

/* Memory to store a group name */
CHAR group_name[20];

/* Add a new group to the database. */
status = IKE_Add_Group("ike_test_group");

if(status == NU_SUCCESS)
{
```

```
/* Add an interface to the group. Assuming that an
 * interface by this name has been initialized already.
 */
status = IKE_Add_To_Group("ike_test_group", "eth0");

if(status == NU_SUCCESS)
{
    /* Now try to find the group containing "eth0 ". */
    status = IKE_Get_Group("eth0", group_name,
                          sizeof(group_name));

    if(status == NU_SUCCESS)
    {
        /* 'group_name' should contain the group name
         * of the matching group. This should be
         * "ike_test_group".
         */
    }
}
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Get\_Group\_Opt

This function returns an IKE group value as specified by the optname parameter.

### Usage

```
STATUS IKE_Get_Group_Opt (CHAR *group_name,  
                          INT  optname,  
                          VOID *optval,  
                          INT  *optlen);
```

### Arguments

- **group\_name**  
Pointer to the name of the group.
- **optname**  
Specifies a group option that is to be found. Valid options values are listed in the [Table 8-13](#).
- **optval**  
Pointer to a location where the option value is placed on a successful request. The data type of the location pointed to by this parameter is specified in the options [Table 8-13](#).
- **optlen**  
Contains the total length, in bytes, of memory pointed to by optval. On return, contains the number of bytes that have been written to optval.

Valid values for option name and value parameters are shown in [Table 8-13](#):

**Table 8-13. IKE\_Get\_Group\_Opt**

Option Name	Option Data Type	Description
IKE_IS_GROUP	None	Function returns NU_SUCCESS if the specified group exists. Option value and length parameters should be NU_NULL.
IKE_TOTAL_POLICIES	UINT32	Total number of policies in the IKE group.
IKE_NEXT_GROUP	CHAR[]	Name of the next group is returned. If no next group exists, then the first group's name is returned.

### Return Values

- **NU\_SUCCESS**  
The request was successful.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.

- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_NOT\_FOUND**  
The specified IKE group was not found.
- **IKE\_LENGTH\_IS\_SHORT**  
Indicates that the value to be returned required more memory than was passed. The number of bytes that were required to be written is returned in `opt_len`.

## Example

```
/* Local variable to store the return status */
STATUS status;

/* Variables to store the option values */
CHAR   group_name[20];
UINT32 total_policies;

/* Variable to store the option length. */
INT     opt_length;

/* First check whether the group exists. */
status = IKE_Get_Group_Opt("ike_test_group", IKE_IS_GROUP,
                           NU_NULL, NU_NULL);

if(status == NU_SUCCESS)
{
    /* The group exists, so now check how many policies it
     * contains. Set the option length accordingly.
     */
    opt_length = sizeof(total_policies);

    status = IKE_Get_Group_Opt("ike_test_group",
                               IKE_TOTAL_POLICIES,
                               &total_policies, &opt_length);

    if(status == NU_SUCCESS)
    {
        /* Here, 'total_policies' contains the total number
         * of policies in the group.
         */
    }

    /* Get the name of the next group. */
    opt_length = sizeof(group_name);

    status = IKE_Get_Group_Opt("ike_test_group",
                               IKE_NEXT_GROUP,
                               &group_name, &opt_length);

    if(status == NU_SUCCESS)
    {
        /* Here, 'group_name' contains the name of the
         * next group, if any. Otherwise, the name of the
         * first group in the database is returned.
         */
    }
}
```



## Related Topics

[IKE API Functions](#)

## IKE\_Get\_Policy\_Index

This function finds the first matching policy corresponding to the passed selector. The selectors are matched using the same method used to match policies for an IKE exchange. The unique identifier for the matched policy is returned.

### Usage

```
STATUS IKE_Get_Policy_Index (CHAR          *group_name,  
                             IKE_POLICY_SELECTOR *selector,  
                             UINT32          *return_index);
```

### Arguments

- **group\_name**  
Pointer to the group name for which the policy is searched.
- **selector**  
Pointer selector to be used to match a policy. The [IKE\\_POLICY\\_SELECTOR/IKE\\_IDENTIFIER](#) structure is defined in the [IKE Data Structures](#) section.
- **return\_index**  
On return, this variable contains the unique identifier for the matched policy.

### Return Values

- **NU\_SUCCESS**  
A matching policy was found.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_NOT\_FOUND**  
No matching policy was found.

### Example

```
/* Local variable to store the return status */  
STATUS          status;  
  
/* Selector to perform the policy look-up */  
IKE_POLICY_SELECTOR select;  
  
/* Variable to store the returned policy identifier */  
UINT32          policy_index;  
  
/* Assuming that the IKE database contains a policy  
 * whose selector specifies a range of IP addresses from  
 * 192.168.0.1 to 192.168.0.20. Construct a single IP-
```

```
    * -based selector which lies within this range so that the
    * range-based policy could be matched.
    */
select.ike_type = IKE_IPV4;

select.ike_addr.ike_ip.ike_addr1[0] = 192;
select.ike_addr.ike_ip.ike_addr1[1] = 168;
select.ike_addr.ike_ip.ike_addr1[2] = 0;
select.ike_addr.ike_ip.ike_addr1[3] = 15;

/* Try to find a matching policy. */
status = IKE_Get_Policy_Index("ike_test_group", &select,
                             &policy_index);

if(status == NU_SUCCESS)
{
    /* A matching policy was found. 'policy_index' would now
    * contain the index of the matched policy.
    */
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Get\_Policy\_Opt

This function gets a policy value as specified by optname. See “[IKE Policy](#)” on page 1340 for more information on IKE policies.

### Usage

```
STATUS IKE_Get_Policy_Opt (CHAR    *group_name,  
                          UINT32  index,  
                          INT     optname,  
                          VOID    *optval,  
                          INT     *optlen);
```

### Arguments

- **group\_name**  
Pointer to the group name whose policy values you want to access.
- **index**  
Unique identifier of the policy.
- **optname**  
Specifies a policy option to be looked up. Valid values are listed in the options [Table 8-14](#). Refer to the “[IKE Policy](#)” on page 1340 for more information about these options.
- **optval**  
Pointer to a location where the option value is placed on a successful request. The data type of the location pointed to by this parameter is specified in the options [Table 8-14](#).
- **optlen**  
Contains the total length, in bytes, of memory pointed to by optval. On return, contains the number of bytes that have been written to optval.

Valid values for option name and value parameters are shown in [Table 8-14](#):

**Table 8-14. IKE\_Get\_Policy\_Opt**

Option Name	Option Data Type	Description
IKE_IS_POLICY	None	Function returns NU_SUCCESS if the specified group exists. Option value and length parameters should be NU_NULL.
IKE_SELECTOR	<a href="#">IKE_POLICY_SELECTOR/IKE_IDENTIFIER</a>	Selector of the policy.
IKE_IDS	<a href="#">IKE_IPS_ID[]</a>	List of IDs allowed in Phase 2.
IKE_PHASE1_XCHG	UINT8	Flags indicating allowable Phase 1 exchanges.

**Table 8-14. IKE\_Get\_Policy\_Opt (cont.)**

Option Name	Option Data Type	Description
IKE_PHASE2_XCHG	UINT8	Flags indicating allowable Phase 2 exchanges.
IKE_XCHG1_ATTRIBS	<a href="#">IKE_ATTRIB[]</a>	List of Phase 1 exchange attributes.
IKE_NEXT_POLICY	UINT32	Unique identifier of next policy, if it exists. Otherwise, identifier of the first policy is returned.
IKE_FLAGS	UINT8	Flags variable of the policy.
IKE2_FLAGS_OPT	UINT8	IKEv2 flags set in the policy.
IKE2_SA_TIMEOUT_OUT	UINT32	Timeout value for IKE SA.
IKE2_VERSION_OPT	UINT8	Returns the version of IKE being used for this policy.

### Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_NOT\_FOUND**  
The group was not found, or there is no matching policy in the specified group.
- **IKE\_LENGTH\_IS\_SHORT**  
Indicates that the value to be returned required more memory than was passed. The number of bytes that were required to be written is returned in optlen.

### Example

```

/* Local variable to store the return status */
STATUS          status;

/* Variables to store the option values */
IKE_POLICY_SELECTOR select;
IKE_IPS_ID        ids[5];
IKE_ATTRIB        attribs[3];

/* Variable to store the option length */
INT              opt_length;

```

```
/* Assuming that POLICY_INDEX is a variable which contains
 * a valid policy index.
 */

/* Set option length to length of a selector. */
opt_length = sizeof(select);

/* Get the policy selector option. */
status = IKE_Get_Policy_Opt("ike_test_group", POLICY_INDEX,
                           IKE_SELECTOR, &select, &opt_length);

if(status == NU_SUCCESS)
{
    /* The 'select' variable now contains the selector of
     * the policy.
     */
}

/* Set option length to the length of five IDs. */
opt_length = sizeof(ids);

/* Get the allowed IDs option. */
status = IKE_Get_Policy_Opt("ike_test_group", POLICY_INDEX,
                           IKE_IDS, &ids, &opt_length);

if(status == NU_SUCCESS)
{
    /* The 'ids' array now contains a list of allowed
     * IDs. 'opt_length / sizeof(IKE_IPS_ID)' is equal to
     * the number of valid elements in this array.
     */
}

/* Set option length to the length of three sets of Phase 1
 * attributes.
 */
opt_length = sizeof(attrs);

/* Get the policy attributes option. */
status = IKE_Get_Policy_Opt("ike_test_group", POLICY_INDEX,
                           IKE_XCHG1_ATTRS, &attrs,
                           &opt_length);

if(status == NU_SUCCESS)
{
    /* The 'attrs' array now contains a list of Phase 1
     * attributes. 'opt_length / sizeof(IKE_ATTRIB)' is equal
     * to the number of valid elements in this array.
     */
}
```

## Related Topics

### [IKE API Functions](#)

## IKE\_Get\_Preshared\_Key\_Index

This function looks up a pre-shared key based on an identifier and returns the unique index of the matching pre-shared key. The method used to look up the pre-shared key based on the identifier is the same as that used for pre-shared key lookup during an IKE exchange. See “[IKE Pre-shared Keys](#)” on page 1409 for more information on pre-shared keys.

### Note



This API is specific to IKEv1 and should not be used for IKEv2. The pre-shared keys for IKEv2 are specified within the policy structure.

## Usage

```
STATUS IKE_Get_Preshared_Key_Index (IKE_IDENTIFIER *id,  
                                   UINT16         *return_index);
```

## Arguments

- id

Pointer to an identifier used for looking up a pre-shared key. This identifier is matched logically to check whether it lies within the range specified by the identifiers of the pre-shared keys within the database. A wildcard identifier matches all pre-shared keys, but only the first match is always returned. The two structures

[IKE\\_POLICY\\_SELECTOR/IKE\\_IDENTIFIER](#) are identical and are described in the [IKE Data Structures](#) section.

- return\_index

On return, this contains a unique index of the matching pre-shared key, if found.

## Return Values

- NU\_SUCCESS  
The request was successful.
- NU\_TIMEOUT  
Timed out waiting for a resource.
- IKE\_INVALID\_PARAMS  
One of the input parameters is invalid.
- IKE\_NOT\_FOUND  
No pre-shared key matches the search criteria.

## Example

```
/* Local variable to store the return status */  
STATUS status;  
  
/* Variable to store the unique index */  
UINT16 psk_index;
```

```
/* Variable to specify the identifier to be searched */
IKE_IDENTIFIER id;

/* Set identifier to wildcard, to make it match all keys. */
id.ike_type = IKE_WILDCARD;

/* Loop until all pre-shared keys are removed from the
 * database.
 */
do
{
    status = IKE_Get_Preshared_Key_Index(&id, &psk_index);

    if(status == NU_SUCCESS)
    {
        /* Remove the pre-shared key which was found. This
         * function should not return an error under normal
         * operation since we can be sure that the pre-shared
         * key exists in the database.
         */
        IKE_Remove_Preshared_Key(psk_index);
    }
} while(status == NU_SUCCESS)

/* When control reaches here, all pre-shared keys have
 * been removed from the database.
 */
```

## Related Topics

### [IKE API Functions](#)



## IKE\_Initiate

This function explicitly initiates an IKE exchange. Valid IKE and IPsec policies must be present in the database before an exchange is initiated. This is an advanced API function. Most applications that depend on IKE do not require direct use of this function. See “[Exchange Invocation](#)” on page 1427 for more information on initiating an IKE exchange.

### Usage

```
STATUS IKE_Initiate (IKE_INITIATE_REQ *request);
```

### Arguments

- request

Pointer to an IKE exchange initiation request. The [IKE\\_INITIATE\\_REQ](#) structure is defined in the [IKE Data Structures](#) section. IKE creates a copy of this structure for use during the exchange, so it could be specified in a local variable. It does not need to stay allocated for as long as the exchange continues, in case of a non-blocking request.

### Return Values

- NU\_SUCCESS  
Policy was successfully removed.
- NU\_TIMEOUT  
Timed out waiting for a resource.
- NU\_NO\_MEMORY  
Not enough memory to service the request.
- IPSEC\_NOT\_FOUND  
IPsec group was not found, or there is no Nucleus IPsec policy corresponding to the selector specified in the request.
- IKE\_INVALID\_STATE  
The IKE service is not running.
- IKE\_INDEX\_NOT\_FOUND  
Too many blocking IKE exchanges in progress already. The [IKE\\_MAX\\_WAIT\\_EVENTS](#) configuration macro defines this limit.
- IKE\_INVALID\_PARAMS  
One of the input parameters is invalid.
- IKE\_NOT\_FOUND  
IKE policy not found for the destination address.
- IKE\_UNALLOWED\_XCHG  
Exchange not allowed by IKE policy.

- **IKE\_UNALLOWED\_XCHG2**  
Exchange not allowed by Nucleus IPsec policy.
- **Exchange Status**  
If the initiate request is blocking, the final status of the IKE exchange is returned. If the request is non-blocking, then the status of transmission of the first exchange message is returned. See "Appendix A" for a complete list of exchange status codes.

## Description

Note that this function only takes a single security protocol for negotiation, but if the requested protocol is part of a larger protocol suite, then all members of the suite are negotiated. For example, if negotiation of the AH protocol is requested and the IPsec policy specifies application of AH + ESP, then an IPsec SA for ESP would also be established with this request.

---

### Note



Nucleus IPsec implicitly initiates an IKE exchange internally if an outgoing packet requires security, but a required SA is missing. This API is used when an SA is to be established before its need actually arises.

---

## Example

### Using Pre-shared Keys

```
/* Local variable to store the return status */
STATUS    status;

/* Local variable to store the new policy */
IKE_POLICY    policy;

/* Phase 2 ID allowed by the policy */
IKE_IPS_ID    phase2_id;

/* Alternative Phase 1 attributes to be negotiated in
 * phase 1.
 */
IKE_ATTRIB    attribs[2];

/* Unique identifier of the policy */
UINT32    policy_index = 0;

/* Assuming that the local IP address is 192.168.0.1 */
UINT8    local_ip_addr[] = { 192, 168, 0, 1 };

/* Assuming that the remote IP address is 192.168.0.20 */
UINT8    foreign_ip_addr[] = { 192, 168, 0, 20 };

/* Set the selector so that it selects ALL addresses. */
policy.ike_select.ike_type = IKE_WILDCARD;

/* Set the local Identifier. */
policy.ike_my_id.ike_type = IKE_IPV4;

memcpy(policy.ike_my_id.ike_addr.ike_ip.ike_addr1,
```

```
        local_ip_addr, IP_ADDR_LEN);

/* Set the remote Identifier. */
policy.ike_peers_id.ike_type = IKE_IPV4;

memcpy(policy.ike_peers_id.ike_addr.ike_ip.ike_addr1,
        foreign_ip_addr, IP_ADDR_LEN);

/* Allow all possible modes in phase 1. */
policy.ike_phase1_xchg = (IKE_XCHG_AGGR_FLAG |
                          IKE_XCHG_MAIN_FLAG |
                          IKE_XCHG_INFO_FLAG);

/* Allow all possible modes in phase 2. */
policy.ike_phase2_xchg = (IKE_XCHG_QUICK_FLAG |
                          IKE_XCHG_INFO_FLAG);

/* Set the policy flags to zero. */
policy.ike_flags = 0;

/* Initialize first set of Phase 1 exchange attributes. */
attribs[0].ike_auth_method      = IKE_PSK;
attribs[0].ike_encryption_algo   = IKE_DES;
attribs[0].ike_key_len          = IKE_WILDCARD;
attribs[0].ike_hash_algo        = IKE_MD5;
attribs[0].ike_group_desc       = IKE_GROUP_MODP_768;
attribs[0].ike_sa_lifetime.ike_no_of_secs =
                                28800;

/* Pre_Shared Key for the policy attrib */
attribs[0].ike2_psk_local = "IKETEST12345678!";
attribs[0].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[0].ike2_psk_remote = "IKETEST12345678!";
attribs[0].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Initialize the second set of Phase 1 exchange attributes. */
attribs[1].ike_auth_method      = IKE_PSK;
attribs[1].ike_encryption_algo   = IKE_BLOWFISH;
attribs[1].ike_hash_algo        = IKE_SHA1;
attribs[1].ike_key_len          = 16;
attribs[1].ike_group_desc       = IKE_GROUP_MODP_768;

/* Pre_Shared Key for the policy attrib */
attribs[1].ike2_psk_local = "IKETEST12345678!";
attribs[1].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[1].ike2_psk_remote = "IKETEST12345678!";
attribs[1].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the above exchange attributes in the policy. */
policy.ike_xchg1_attribs_no = 2;
policy.ike_xchg1_attribs    = attribs;

/* Allow all phase 2 exchanges for all protocols. */
phase2_id.ike_protocol_id = IKE_WILDCARD;
phase2_id.ike_port        = IKE_WILDCARD;

/* Set phase 2 protocols/ports allowed by the IKE policy. This could
 * be an array, but only a single wildcard entry would be enough.
 */
```

```
policy.ike_ids      = &phase2_id;
policy.ike_ids_no = 1;

/* Create this policy. Policy index will be returned. */
status = IKE_Add_Policy("ike_test_group", &policy, &policy_index);

if(status == NU_SUCCESS)
{
    /* Successfully added IKE policy to the group. */
}
```

### Using Digital Signatures

```
/* Local variable to store the return status */
STATUS      status;

/* Local variable to store the new policy */
IKE_POLICY  policy;

/* Phase 2 ID allowed by the policy */
IKE_IPS_ID  phase2_id;

/* Alternative Phase 1 attributes to be negotiated in
 * phase 1
 */
IKE_ATTRIB  attribs[1];

/* Unique identifier of the policy. */
UINT32      policy_index = 0;

/* Assuming that the local IP address is 192.168.0.1 */
UINT8       local_ip_addr[] = { 192, 168, 0, 1 };

/* Assuming that the remote IP address is 192.168.0.20 */
UINT8       foreign_ip_addr[] = { 192, 168, 0, 20 };

/* Set the selector so that it selects ALL addresses. */
policy.ike_select.ike_type = IKE_WILDCARD;

/* Set the local Identifier. When using IKE_IPV4 as Identifier
 * type, the local certificate specified should have similar
 * value in the SubjectAltName field otherwise the certificate
 * verification will fail on the peer side.
 */
policy.ike_my_id.ike_type = IKE_IPV4;

memcpy(policy.ike_my_id.ike_addr.ike_ip.ike_addr1,
       local_ip_addr, IP_ADDR_LEN);

/* Set the remote Identifier. (See comment above before
 * setting your own identification. The same case applies here.
 * Different ID types here and in the certificate cause
 * the certificate verification to fail.
 */
policy.ike_peers_id.ike_type = IKE_IPV4;

memcpy(policy.ike_peers_id.ike_addr.ike_ip.ike_addr1,
       foreign_ip_addr, IP_ADDR_LEN);
```

```
/* Allow all possible modes in phase 1. */
policy.ike_phase1_xchg = (IKE_XCHG_AGGR_FLAG |
                          IKE_XCHG_MAIN_FLAG |
                          IKE_XCHG_INFO_FLAG);

/* Allow all possible modes in phase 2. */
policy.ike_phase2_xchg = (IKE_XCHG_QUICK_FLAG |
                          IKE_XCHG_INFO_FLAG);

/* Set policy flags to zero. */
policy.ike_flags = IKE_CA_IN_CERTREQ;

/* Initialize first set of Phase 1 exchange attributes. */
attribs[0].ike_auth_method = IKE_RSA;
attribs[0].ike_encryption_algo = IKE_DES;
attribs[0].ike_key_len = IKE_WILDCARD;
attribs[0].ike_hash_algo = IKE_MD5;
attribs[0].ike_group_desc = IKE_GROUP_MODP_768;
attribs[0].ike_sa_lifetime.ike_no_of_secs = IPSEC_WILDCARD;
attribs[0].ike_local_cert_file = "cacert.der";
attribs[0].ike_local_key_file = "keyout.der";
attribs[0].ike_ca_cert_file = "sim_cert.der";
attribs[0].ike_cert_encoding = IKE_X509_FILETYPE_ASN1;

/* Pre_Shared Key for the policy attrib */
attribs[1].ike2_psk_local = "IKETEST12345678!";
attribs[1].ike2_psk_local_len = strlen("IKETEST12345678!");
attribs[1].ike2_psk_remote = "IKETEST12345678!";
attribs[1].ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the above exchange attributes in the policy. */
policy.ike_xchgl_attribs_no = 1;
policy.ike_xchgl_attribs = attribs;

/* Allow all phase 2 exchanges for all protocols. */
phase2_id.ike_protocol_id = IKE_WILDCARD;
phase2_id.ike_port = IKE_WILDCARD;

/* Set phase 2 protocols/ports allowed by the IKE
 * policy. This could be an array, but only a single wildcard
 * entry would be enough.
 */
policy.ike_ids = &phase2_ids;
policy.ike_ids_no = 1;

/* Create this policy. Policy index is returned. */
status = IKE_Add_Policy("ike_test_group", &policy,
                       &policy_index);

if(status == NU_SUCCESS)
{
    /* Successfully added IKE policy to the group. */
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Remove\_From\_Group

This function removes a network interface from the IKE group with which it is associated.

### Usage

```
STATUS IKE_Remove_From_Group (CHAR *interface_name);
```

### Arguments

- `interface_name`  
Pointer to the name of the interface to be removed from the group.

### Return Values

- `NU_SUCCESS`  
The request was successful.
- `NU_TIMEOUT`  
Timed out waiting for a resource.
- `IKE_INVALID_PARAMS`  
The input parameter is invalid.
- `IKE_NOT_FOUND`  
The specified interface was not found, or it is not associated with any IKE group.

### Example

```
/* Local variable to store the return status */
STATUS status;

/* Add a new group to the database. */
status = IKE_Add_Group("ike_test_group");

if(status == NU_SUCCESS)
{
    /* Add a network interface to the group. */
    status = IKE_Add_To_Group("ike_test_group", "eth0");

    if(status == NU_SUCCESS)
    {
        /* Remove the interface from its associated group. */
        status = IKE_Remove_From_Group("eth0");
    }
}
```

### Related Topics

[IKE API Functions](#)

## IKE\_Remove\_Group

This function removes an IKE group from the database. All associated interfaces are removed from the group and all policies of the group are also removed.

### Usage

```
STATUS IKE_Remove_Group (CHAR *group_name);
```

### Arguments

- group\_name  
Pointer to name of the group.

### Return Values

- NU\_SUCCESS  
The request was successful.
- NU\_TIMEOUT  
Timed out waiting for a resource.
- IKE\_INVALID\_PARAMS  
The input parameter is invalid.
- IKE\_NOT\_FOUND  
The group was not found in the database.

### Example

```
/* Local variable to store the return status */
STATUS status;

/* Add a new group to the database. */
status = IKE_Add_Group("ike_test_group");

if(status == NU_SUCCESS)
{
    /* Add a network interface to the group. */
    status = IKE_Add_To_Group("ike_test_group", "eth0");

    if(status == NU_SUCCESS)
    {
        /* Remove the group. This would also disassociate
         * the "eth0" interface associated with this group
         * above.
         */
        status = IKE_Remove_Group("ike_test_group");
    }
}
```



## Related Topics

[IKE API Functions](#)

## IKE\_Remove\_Policy

This function removes a policy from the IKE database. Memory that was previously allocated is deallocated.

### Usage

```
STATUS IKE_Remove_Policy (CHAR    *group_name,  
                          UINT32  index);
```

### Arguments

- `group_name`  
Pointer to the group name from which the policy needs to be removed.
- `index`  
Unique identifier for the policy that needs to be removed.

### Return Values

- `NU_SUCCESS`  
Policy was successfully removed.
- `NU_TIMEOUT`  
Timed out waiting for a resource.
- `IKE_INVALID_PARAMS`  
One of the input parameters is invalid.
- `IKE_NOT_FOUND`  
The specified policy was not found.

### Example

```
/* Local variable to store the return status */  
STATUS          status;  
  
/* Selector to perform the policy look-up */  
IKE_POLICY_SELECTOR select;  
  
/* Variable to store the returned policy identifier */  
UINT32          policy_index;  
  
/* Create a wildcard selector which matches all policies. */  
select.ike_type = IKE_WILDCARD;  
  
/* Try to find a matching policy. */  
status = IKE_Get_Policy_Index("ike_test_group", &select,  
                              &policy_index);  
  
if(status == NU_SUCCESS)  
{  
    /* Delete the matching policy. */  
    status = IKE_Remove_Policy("ike_test_group", policy_index);  
}
```

```
    if(status == NU_SUCCESS)
    {
        /* Policy deleted successfully. */
    }
}
```

## Related Topics

[IKE API Functions](#)

## IKE\_Remove\_Preshared\_Key

This function removes a pre-shared key from the database. The pre-shared key is identified by its unique index.

### Note



This API is specific to IKEv1 and should not be used for IKEv2. The pre-shared keys for IKEv2 are specified within the policy structure.

---

### Usage

```
STATUS IKE_Remove_Preshared_Key (UINT16 index);
```

### Arguments

- **index**  
Unique index of the pre-shared key to be removed from the database.

### Return Values

- **NU\_SUCCESS**  
The request was successful.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
The input parameter is invalid.
- **IKE\_NOT\_FOUND**  
No pre-shared key matches the specified index.

### Example

```
/* Local variable to store the return status */
STATUS status;

/* Variable to store the unique index.*/
UINT16 psk_index;

/* Variable to specify the identifier to be searched.*/
IKE_IDENTIFIER id;

/* Set the identifier to a wildcard, to make it match all keys. */
id.ike_type = IKE_WILDCARD;

/* Loop until all pre-shared keys are removed from the
 * database.
 */
do
{
    status = IKE_Get_Preshared_Key_Index(&id, &psk_index);
```

```
    if(status == NU_SUCCESS)
    {
        /* Remove the pre-shared key which was found. This
        * function should not return an error under normal
        * operation since you can be sure that the pre-shared
        * key exists in the database.
        */
        IKE_Remove_Preshared_Key(psk_index);
    }
} while(status == NU_SUCCESS)

/* When control reaches here, all pre-shared keys would have
* been removed from the database.
*/
```

## Related Topics

[IKE API Functions](#)

## IKE\_Set\_Policy\_Opt

This function sets a policy value as specified by optname. See “IKE Policy” on page 1340 for more information on IKE policies.

### Usage

```
STATUS IKE_Set_Policy_Opt (CHAR    *group_name,  
                          UINT32  index,  
                          INT      optname,  
                          VOID     *optval,  
                          INT      optlen);
```

### Arguments

- **group\_name**  
Pointer to the group name whose policy values are being set.
- **index**  
Unique identifier of the policy.
- **optname**  
Specifies a policy option which is to be set. Valid values are listed in [Table 8-15](#). Refer to the description of [IKE\\_POLICY](#) for an understanding of these options.
- **optval**  
Pointer to a location from where the option value is copied on a successful request. The data type of the location pointed to by this parameter is specified in [Table 8-15](#).
- **optlen**  
Length, in number of bytes, of the value that is to be copied on a successful request.

Valid values for option name and value parameters are shown in [Table 8-15](#):

**Table 8-15. IKE\_Set\_Policy\_Opt**

Option Name	Option Data Type	Description
IKE_PHASE1_XCHG	UINT8	Flags indicating allowable Phase 1 exchanges.
IKE_PHASE2_XCHG	UINT8	Flags indicating allowable Phase 2 exchanges.
IKE_FLAGS	UINT8	Flags variable of the policy.
IKE2_FLAGS_OPT	UINT8	IKEv2 flags set in the policy.
IKE2_SA_TIMEOUT_OPT	UINT32	Returns the version of IKE being used for this policy.

## Return Values

- **NU\_SUCCESS**  
The request was successfully executed.
- **NU\_TIMEOUT**  
Timed out waiting for a resource.
- **IKE\_INVALID\_PARAMS**  
One of the input parameters is invalid.
- **IKE\_INVALID\_LENGTH**  
Indicates that the length of the option is invalid.
- **IKE\_NOT\_FOUND**  
The group was not found, or there is no matching policy in the specified group.

## Example

```
/* Local variable to store the return status */
STATUS status;

/* Variables used to set option values */
UINT8 phase1_flags;
UINT8 policy_flags;

/* Variable to store the option length */
INT opt_length;

/* Assuming that POLICY_INDEX is a variable which contains
 * a valid policy index.
 */

/* Set the phase 1 exchange flags option to allow only
 * Aggressive and Informational Mode in phase 1.
 */
phase1_flags = (IKE_XCHG_AGGR_FLAG | IKE_XCHG_INFO_FLAG);
/* Set option length. */
opt_length = sizeof(phase1_flags);

/* Set the phase 1 exchange flags option. */
status = IKE_Set_Policy_Opt("ike_test_group", POLICY_INDEX,
                           IKE_PHASE1_XCHG, &phase1_flags,
                           &opt_length);

if(status == NU_SUCCESS)
{
    /* Policy has been successfully updated to use
     * Aggressive and Informational Mode in phase 1.
     */
}

/* Set policy flags to enable strict ID verification. */
policy_flags = IKE_VERIFY_ID;

/* Set option length. */
```

```
opt_length = sizeof(policy_flags);

/* Set the policy flags option. */
status = IKE_Set_Policy_Opt("ike_test_group", POLICY_INDEX,
                             IKE_FLAGS, &policy_flags,
                             &opt_length);

if(status == NU_SUCCESS)
{
    /* Policy has been successfully updated to strictly verify
     * remote host identity in phase 1.
     */
}
```

## Related Topics

[IKE API Functions](#)



## IKE\_Shutdown

This function shuts down IKE. It removes all IKE SAs, policies, and groups and terminates the service tasks.

---

**Note**

IKE may take a few seconds to shut down. This function blocks until the shutdown is complete.

---

### Usage

```
STATUS IKE_Shutdown (VOID);
```

### Return Values

- **NU\_SUCCESS**  
IKE shutdown was successful.
- **IKE\_INVALID\_STATE**  
The IKE service is not running. It must be running when a shutdown request is issued.

### Example

```
/* Local variable to store the return status */
STATUS status;

/* Assuming that IKE is already running, issue a
 * shutdown request. This might take a few seconds.
 */
status = IKE_Shutdown();

if(status == NU_SUCCESS)
{
    /* IKE shutdown was successful. */
}
```

### Related Topics

[IKE API Functions](#)

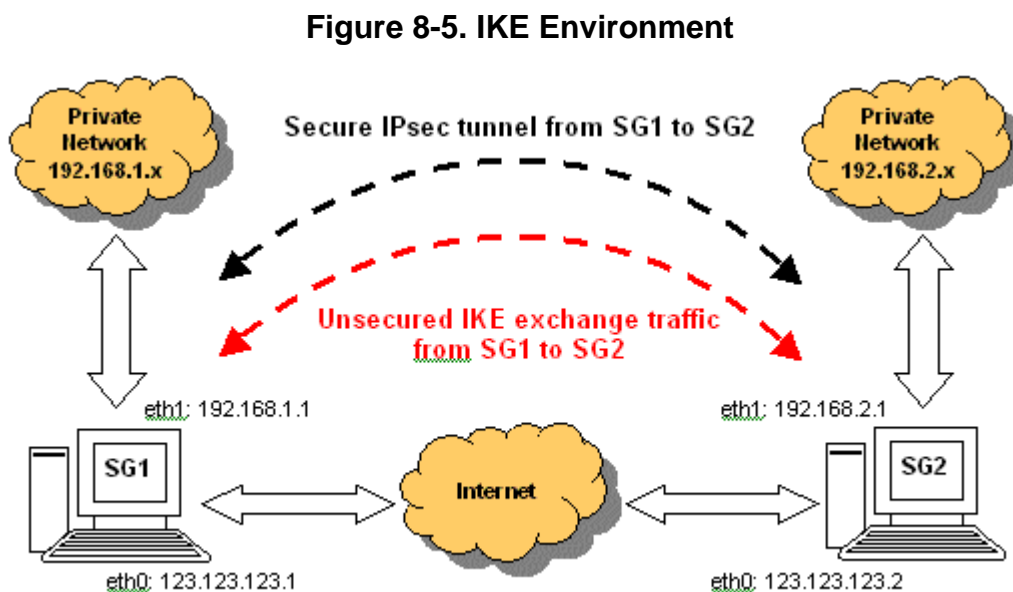
## IKE API Usage Example

This section provides an example usage of IKE services. An environment is first described and, then, IKE services are used to create that environment. Although all of the APIs are not used, this section gives enough details to provide the essence of IKE services.

### IKE Environment

There are two security gateways, SG1 and SG2, which connect two private networks over a public network, such as the Internet. Nucleus IPsec is being used on SG1. Both security gateways have two physical interfaces: eth0 (interface connected to the public network) and eth1 (interface connected to the private network).

Figure 8-5 illustrates this configuration:



Both security gateways must have similar policies. Two IPsec policies are required. The first one is used to bypass security on all IKE exchange traffic. The parameters of this policy are listed in Table 8-16 :

**Table 8-16. Bypass Security Policies**

Policy Option	Value
IPsec group name	“ipsec_test_group”
Policy action	Bypass
Policy direction	Dual asynchronous
Security Mode	Transport Mode

**Table 8-16. Bypass Security Policies (cont.)**

Policy Option	Value
Security Protocol	None
Transport Protocol	UDP

A second IPsec policy is required to secure the private network's traffic between the security gateways. Parameters of this policy are listed in [Table 8-17](#):

**Table 8-17. Network Traffic Security Policy**

Policy Option	Value
IPsec group name	"ipsec_test_group"
Policy action	Apply
Policy direction	Dual asynchronous
Security Mode	Tunnel Mode
Security Protocol	ESP
ESP Encryption	DES-CBC
ESP Authentication	HMAC-MD5-96
Transport Protocol	Wildcard
IPsec SA lifetime	172800 seconds
Perfect Forward Secrecy	Disabled

Parameters for the IKE policy are listed [Table 8-18](#):

**Table 8-18. IKE Policy**

Policy Option	Value
IKE group name	"ike_test_group"
Allowable phase 1 modes	Aggressive Mode, Informational Mode
Allowable phase 2 modes	Quick Mode, Informational Mode
Authentication Method	Pre-shared key ("topsecretkey")
Oakley group	First Oakley Group (MODP-768)
Encryption algorithm	DES
Authentication algorithm	MD5
IKE SA lifetime	28800 seconds

### Note



The authentication method used in this example is Pre-shared Keys. Authentication can also be done using digital signatures. For that purpose, both ends should have their own CAs certificates. In the following code samples, see [“IKE Policy”](#) on page 1340 for information on how to set up authentication by digital certificates. The rest of the scenario remains as it is.

---

## IKE Code Samples

This section provides code samples to implement the policies previously discussed. It is assumed that the two physical interfaces have already been initialized. Refer to the [NET](#) chapter for information on initializing physical interfaces.

eth0 is the name assigned to the first network interface, and eth1 is the name assigned to the second network interface during initialization. All of the following settings apply to SG1, although a similar policy must also exist on SG2.

### Nucleus IPsec Groups

The first step in creation of the Nucleus IPsec policies is to define the group. In this case, there would be a single group that contains a single interface, eth0.

```
/* Create a new group with the name "ipsec_test_group". */
if(IPSEC_Add_Group("ipsec_test_group") == NU_SUCCESS)
{
    /* Add the network interface connected to the public
     * network to this group.
     */
    IPSEC_Add_To_Group("ipsec_test_group", "eth0");
}
```

### Nucleus IPsec Policies

The following code adds a Nucleus IPsec policy to bypass the IKE exchange traffic. IKE exchange uses a SA of its own, so it should not be protected by IPsec. This should be the first policy added to ensure that it gets matched first and does not get overridden by another policy.

```
/* Local variables used to construct the policy */
IPSEC_POLICY bypass_pol;
UINT32      bypass_pol_index;

/* Set the policy selector so that it selects all UDP
 * traffic. */
bypass_pol.ipsec_select.ipsec_transport_protocol = IPPROTO_UDP;
bypass_pol.ipsec_select.ipsec_source_type       = IPSEC_WILDCARD;
bypass_pol.ipsec_select.ipsec_source_port       = IPSEC_WILDCARD;
bypass_pol.ipsec_select.ipsec_dest_type         = IPSEC_WILDCARD;
bypass_pol.ipsec_select.ipsec_destination_port  = IPSEC_WILDCARD;

/* Set policy to apply to both inbound and outbound packets. */
```

```
bypass_pol.ipsec_flags = (IPSEC_BY_PASS |
                          IPSEC_DUAL_ASYNCHRONOUS);

/* Set lifetime of bundles. This would be unused for a bypass
 * policy. */
bypass_pol.ipsec_bundles.ipsec_lifetime = 172800;

/* Security protocols must not be specified for bypass
 * policies. */
bypass_pol.ipsec_security      = NU_NULL;
bypass_pol.ipsec_security_size = 0;

/* Create this policy. The policy index is returned. */
if(IPSEC_Add_Policy("ipsec_test_group", &bypass_pol,
                  &bypass_pol_index) != NU_SUCCESS)
{
    /* Unable to create policy. */
}
```

The following code adds a Nucleus IPsec policy to secure the private network traffic between the two security gateways.

```
/* Local variables used to construct the policy. */
IPSEC_POLICY      apply_pol;
UINT32            apply_pol_index;
IPSEC_SECURITY_PROTOCOL security;

/* Set the policy selector so that it selects source as
 * the local private network (192.168.1.x) and destination as
 * the remote private network (192.168.2.x) at the other end of
 * the IPsec tunnel.
 */
apply_pol.ipsec_select.ipsec_source_port      = IPSEC_WILDCARD;
apply_pol.ipsec_select.ipsec_destination_port = IPSEC_WILDCARD;
apply_pol.ipsec_select.ipsec_transport_protocol =
                                          IPSEC_WILDCARD;

apply_pol.ipsec_select.ipsec_source_type = (IPSEC_IPV4 |
                                           IPSEC_SUBNET_IP);
apply_pol.ipsec_select.ipsec_dest_type  = (IPSEC_IPV4 |
                                           IPSEC_SUBNET_IP);

/* Set the local private network's subnet in the selector. */
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr[0] = 192;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr[1] = 168;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr[2] = 1;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr[3] = 0;

apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr2[0] = 255;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr2[1] = 255;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr2[2] = 255;
apply_pol.ipsec_select.ipsec_source_ip.ipsec_addr2[3] = 0;

/* Set the remote private network's subnet in the selector. */
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr[0] = 192;
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr[1] = 168;
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr[2] = 2;
```

```
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr[3] = 0;

apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr2[0] = 255;
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr2[1] = 255;
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr2[2] = 255;
apply_pol.ipsec_select.ipsec_dest_ip.ipsec_addr2[3] = 0;

/* Set the policy to apply to both inbound and outbound packets. */
apply_pol.ipsec_flags = (IPSEC_APPLY | IPSEC_DUAL_ASYNCHRONOUS);

/* Set the SA lifetime maximum limit. */
apply_pol.ipsec_sa_max_lifetime.ipsec_expiry_action = 0;
apply_pol.ipsec_sa_max_lifetime.ipsec_no_of_secs = 172800;

/* Set the lifetime of bundles. */
apply_pol.ipsec_bundles.ipsec_lifetime = 172800;

/* Set the PFS group to none. */
/* Specify the phase 2 Diffie-Hellman group. The following are
 * commonly used values:
 * - IKE_GROUP_NONE
 * - IKE_GROUP_MODP_768
 * - IKE_GROUP_MODP_1024
 */

apply_pol.ipsec_pfs_group_desc = IKE_GROUP_NONE;

/* Set the security item in the policy. Only a single security
 * protocol would be used.
 */
apply_pol.ipsec_security = &security;
apply_pol.ipsec_security_size = 1;

/* Initialize the policy's IPsec security protocol. */
security.ipsec_security_mode = IPSEC_TUNNEL_MODE;
security.ipsec_protocol = IPSEC_ESP;
security.ipsec_sa_derivation = IPSEC_VALUE_FROM_POLICY;
security.ipsec_auth_algo = IPSEC_HMAC_MD5_96;
security.ipsec_encryption_algo = IPSEC_DES_CBC;

/* Also specify the tunnel end points in the security
 * protocol.
 */
security.ipsec_tunnel_source[0] = 123;
security.ipsec_tunnel_source[1] = 123;
security.ipsec_tunnel_source[2] = 123;
security.ipsec_tunnel_source[3] = 1;

security.ipsec_tunnel_destination[0] = 123;
security.ipsec_tunnel_destination[1] = 123;
security.ipsec_tunnel_destination[2] = 123;
security.ipsec_tunnel_destination[3] = 2;

/* Specify the tunnel address family. */
security.ipsec_flags = IPSEC_IPV4;

/* Create this policy. The policy index is returned. */
```

```
if(IPSEC_Add_Policy("ipsec_test_group", &apply_pol,  
                  &apply_pol_index);
```

## IKE Groups

The first step in creation of the IKE policy is to define a group. In this case, there would be a single group that contains a single interface, eth0.

```
/* Create a new group with the name "ike_test_group". */  
if(IKE_Add_Group("ike_test_group") == NU_SUCCESS)  
{  
    /* Add the network interface connected to the public  
     * network to this group.  
     */  
    IKE_Add_To_Group("ike_test_group", "eth0");  
}
```

The following code can now be used to determine the group for SG1's eth0 interface:

```
/* Local variable used to store the returned group name */  
CHAR group_name[20];  
  
/* Local variable used to pass the total length that can be put  
 * in the group_name variable. On return this contains the  
 * length of the name copied.  
 */  
UINT32 len = 20;  
  
/* Get the group for the "eth0" interface. On return,  
 * 'group_name' should contain "ike_test_group" and 'len'  
 * should be equal to 15.  
 */  
IKE_Get_Group("eth0", group_name, &len);
```

## IKE Pre-shared Keys

The following code adds the pre-shared key used for a key exchange between SG1 and SG2:

```
/* Local variable used to pass information about the pre-shared  
 * key that is to be created  
 */  
IKE_PRESHARED_KEY psk;  
  
/* Local variable used to store the unique index assigned  
 * to the new pre-shared key  
 */  
UINT16 psk_index;  
  
/* Set the pre-shared key identifier to the IP address  
 * of SG2.  
 */  
psk.ike_id.ike_type = IKE_IPV4;  
  
psk.ike_id.ike_addr.ike_ip.ike_addr1[0] = 123;  
psk.ike_id.ike_addr.ike_ip.ike_addr1[1] = 123;  
psk.ike_id.ike_addr.ike_ip.ike_addr1[2] = 123;
```

```
psk.ike_id.ike_addr.ike_ip.ike_addr1[3] = 2;

/* Set the pre-shared key material. Make sure the string
 * terminator is not counted in the key length.
 */
psk.ike_key      = (UINT8 *)"topsecretkey";
psk.ike_key_len  = sizeof("topsecretkey") - 1;

/* Add the pre-shared key to the database. */
if(IKE_Add_Preshared_Key(&psk, &psk_index) != NU_SUCCESS)
{
    /* Unable to add the pre-shared key to the database. */
}
```

## IKE Policy

The following code adds the IKE policy for carrying out a key exchange between SG1 and SG2:

```
/* Local variable used to pass information about the policy that
 * is to be created
 */
IKE_POLICY policy;

/* Local variable used to pass information about the IKE
 * specific attributes
 */
IKE_ATTRIB attrib;

/* Variable that is used to store the index assigned to
 * the newly created policy
 */
UINT32      policy_index;

/* Local variable used to specify the ports and protocols for
 * which Phase 2 SAs can be negotiated
 */
IKE_IPS_ID phase2_id;

/* Set the selector so that it selects all addresses. */
policy.ike_select.ike_type = IKE_WILDCARD;

/* Set the local identifier to IPv4. This is being
 * set to the public IP of SG1.
 */
policy.ike_my_id.ike_type = IKE_IPv4;

/* Set the remote identifier to IPv4. This is being
 * set to the public IP of SG2.
 */
policy.ike_peers_id.ike_type = IKE_IPv4;

/* Copy the local and remote IPv4 addresses into the policy structure. */
memcpy(policy.ike_my_id.ike_addr.ike_ip.ike_addr1, sg1_ip_addr,
IP_ADDR_LEN);
memcpy(policy.ike_peers_id.ike_addr.ike_ip.ike_addr1, sg2_ip_addr,
IP_ADDR_LEN);
```



```

/* Set permissible Phase 1 and 2 exchange modes. */
policy.ike_phase1_xchg = (IKE_XCHG_AGGR_FLAG |
                           IKE_XCHG_INFO_FLAG);
policy.ike_phase2_xchg = (IKE_XCHG_QUICK_FLAG |
                           IKE_XCHG_INFO_FLAG);

/* Initialize phase 1 exchange attributes. */
attrib.ike_auth_method      = IKE_PSK;
/* To use authentication by digital signatures, use IKE_RSA in
 * place of IKE_PSK in the above line as follows:
 * attrib.ike_auth_method      = IKE_RSA;
 */
attrib.ike_encryption_algo   = IKE_DES;
attrib.ike_key_len           = IKE_WILDCARD;
attrib.ike_hash_algo        = IKE_MD5;
attrib.ike_group_desc        = IKE_GROUP_MODP_768;
attrib.ike_sa_lifetime.ike_no_of_secs = 28800;
/* When authenticating using digital signatures, values for the
 * following additional fields must to be specified.
 */
/*
attribs.ike_local_cert_file   = "local_cert_file.der";
attribs.ike_local_key_file    = "local_key_file.der";
attribs.ike_ca_cert_file      = "CA_cert_file.der";
attribs.ike_cert_encoding     = IKE_X509_FILETYPE_ASN1;
*/

/* Pre_Shared Key for the policy attrib */
attribs.ike2_psk_local = "IKETEST12345678!";
attribs.ike2_psk_local_len = strlen("IKETEST12345678!");
attribs.ike2_psk_remote = "IKETEST12345678!";
attribs.ike2_psk_remote_len = strlen("IKETEST12345678!");

/* Set the exchange attributes in the policy. Only
 * a single set of attributes has been defined.
 */
policy.ike_xchg1_attribs      = &attrib;
policy.ike_xchg1_attribs_no = 1;

/* Allow phase 2 exchanges for all ports/protocols. */
phase2_id.ike_protocol_id = IKE_WILDCARD;
phase2_id.ike_port        = IKE_WILDCARD;

/* Set the above phase 2 ID in the policy. Since a wildcard
 * rule is used, only a single ID would be enough for all
 * exchanges.
 */
policy.ike_ids      = &phase2_id;
policy.ike_ids_no = 1;

/* Create this policy. The policy index is returned. */
if(IKE_Add_Policy("ike_test_group", &policy, &policy_index)
    != NU_SUCCESS)
{
    /* Unable to create IKE policy. */
}

```

The following code gets the index for the policy just created. Note that this code can also be used to verify that the policy just created is actually used to process IKE traffic between the two security gateways, and that some other policy will not supersede this.

```
/* Local variable used to store the selection criteria */
IKE_POLICY_SELECTOR selector;

/* Local variable used to store the policy index */
UINT32 policy_index;

/* Set the selector so that it selects packets to an IP
 * in the private network behind SG2.
 */
selector.ike_type = IKE_IPV4;

selector.ike_select.ike_addr.ike_ip.ike_addr1[0] = 192;
selector.ike_select.ike_addr.ike_ip.ike_addr1[1] = 168;
selector.ike_select.ike_addr.ike_ip.ike_addr1[2] = 2;
selector.ike_select.ike_addr.ike_ip.ike_addr1[3] = 30;

/* Get the policy index. Note that this function uses the search
 * criteria that is actually used to find an IKE policy
 * when a request is received. On return, the policy_index
 * should have the same value as when the policy was created.
 * This would verify that this policy is used for requests
 * matching this criteria.
 */
IKE_Get_Policy_Index("ike_test_group", &selector,
                    &policy_index);
```

The following code can be used to traverse all the policies in the "ike\_test\_group" group.

```
/* Local variable used to store the policy_index */
UINT32 policy_index

/* Local variable used to store the index of the first
 * policy
 */
UINT32 first_index;

/* Local variable used to store index of the previous policy. */
UINT32 prev_index;

/* Local variable used to pass the length. On return, this
 * contains the number of bytes written.
 */
INT optlen = sizeof(UINT32);

/* Local variable used to store the selection criteria that is
 * used to get the first policy.
 */
IKE_POLICY_SELECTOR selector;

/* Make the selector match all policies. */
selector.ike_type = IKE_WILDCARD;
```

```
/* Get the first policies index. Since the selection criteria
 * has a wildcard for all fields, you will get the first policy.
 */
if(IKE_Get_Policy_Index("ike_test_group", &selector,
                        &first_index) == NU_SUCCESS)
{
    /* Set the previous policy index to the first policy index
     * in the list to retrieve the next policies.
     */
    prev_index = first_index;

    /* Now go through all the policies. Note that the policy
     * list is not sorted in order of policy indices
     * therefore, the indices returned are not necessarily
     * in ascending order.
     */
    do
    {
        /* Get the next policies index. */
        IKE_Get_Policy_Opt("ike_test_group", prev_index,
                           IKE_NEXT_POLICY, &policy_index,
                           &optlen);

        /* Update the previous policy index to retrieve the
         * subsequent policies from the list.
         */
        prev_index = policy_index;
    } while(first_index != policy_index);
}
```

The following code can be used to update the policy to use Main Mode instead of Aggressive Mode in the Phase 1 exchange.

```
/* Local variable that gives the new exchange flags value
 * that are used.
 */
UINT8 phase1_xchg;

/* Set the new value for the phase 1 exchange flags. */
phase1_xchg = (IKE_XCHG_MAIN_FLAG | IKE_XCHG_INFO_FLAG);

/* Set the new selector values. */
IKE_Set_Policy_Opt("ike_test_group", POLICY_INDEX,
                   IKE_PHASE1_XCHG, &phase1_xchg,
                   sizeof(phase1_xchg));
```

## IKE Customizations

This chapter describes how IKE can be customized to include support for a different set of hash and encryption algorithms.

### Note



Any new hash or encryption algorithm being added to IKE must be first added to Nucleus SSL. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for information on adding new algorithms to this product.

---

## Adding Phase 1 Encryption Algorithm

IKE can be customized to add new encryption algorithms that are negotiated for IKE SAs in Phase 1. Perform the following steps to add support for a new encryption algorithm.

### Procedure

1. Add support for the new encryption algorithm to the Nucleus SSL library used by IKE. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for more information on adding new algorithms to this library. Also, add the corresponding crypto identifier in the *ike\_crypto\_wrappers.h* file.
2. Ensure that the Internet Assigned Numbers Authority (IANA) has assigned a Phase 1 encryption algorithm ID to the algorithm being added. This ID would be defined by an RFC that describes support of the encryption algorithm for IPsec and IKE. Add a macro for this algorithm to the "IKE DOI and other Phase 1 constants" section of *ike\_doi.h*, if it is not already present. For example, if Twofish has been assigned an ID of 12345, then the Twofish macro is added as follows:

```
/* ISAKMP SA attribute values for Encryption Algos */
#define IKE_VAL_DES_CBC 1
#define IKE_VAL_IDEA_CBC 2
#define IKE_VAL_BF_CBC 3
#define IKE_VAL_RC5_R16_B64_CBC 4
#define IKE_VAL_3DES_CBC 5
#define IKE_VAL_CAST_CBC 6
#define IKE_VAL_AES_CBC 7
#define IKE_VAL_TWOFISH_CBC 12345
```

Note that in the previous example, if support for IDEA encryption is being added instead of Twofish, then no modification would be required because `IKE_VAL_IDEA_CBC` has already been defined in the default list of macros.

3. Define the `IKE_INCLUDE_XXX` macro in *os/include/networking/ike\_cfg.h* where XXX is the algorithm name. If this macro is set to `NU_TRUE`, the algorithm is included in the build, and if set to `NU_FALSE`, the algorithm is excluded from the build. The following sample demonstrates adding the include macro for Twofish encryption.

```
/* Encryption Algorithms. These are specific to the
 * phase 1 encryption in an IKE exchange. For
 * configuration of encryption used in IPsec SAs, see
 * the Nucleus IPsec configuration file.
 */
#define IKE_INCLUDE_DES            NU_TRUE /* DES. */
#define IKE_INCLUDE_3DES          NU_TRUE /* 3DES. */
#define IKE_INCLUDE_BLOWFISH      NU_TRUE /* Blowfish. */
#define IKE_INCLUDE_CAST128       NU_TRUE /* CAST-128. */
#define IKE_INCLUDE_TWOFISH       NU_TRUE /* Twofish. */
```

4. Add the algorithm include macro, defined in Step 3, to the `IKE_TOTAL_ENCRYPTION_ALGO` macro. The following is an example for adding the `IKE_INCLUDE_TWOFISH` encryption algorithm identifier.

```
/* This macro calculates the total number of encryption
 * algorithms included in the build.
 */
#define IKE_TOTAL_ENCRYPTION_ALGO  (IKE_INCLUDE_DES      + \
                                     IKE_INCLUDE_3DES      + \
                                     IKE_INCLUDE_BLOWFISH  + \
                                     IKE_INCLUDE_CAST128   + \
                                     IKE_INCLUDE_TWOFISH)
```

5. Define an IKE algorithm identifier and set its value equal to the IANA defined algorithm identifier, defined in Step 2. The identifier macro must be defined in `os/include/networking/ike_cfg.h`. For example, add the identifier for the Twofish algorithm as follows:

```
/* DES algorithm constants */
#if (IKE_INCLUDE_DES == NU_TRUE)
#define IKE_DES            IKE_VAL_DES_CBC
#else
#define IKE_DES            -1
#endif

/* 3DES algorithm constants */
#if (IKE_INCLUDE_3DES == NU_TRUE)
#define IKE_3DES           IKE_VAL_3DES_CBC
#else
#define IKE_3DES           -1
#endif

/* Blowfish algorithm constants */
#if (IKE_INCLUDE_BLOWFISH == NU_TRUE)
#define IKE_BLOWFISH       IKE_VAL_BF_CBC
#else
#define IKE_BLOWFISH       -1
#endif

/* CAST128 algorithm constants */
#if (IKE_INCLUDE_CAST128 == NU_TRUE)
#define IKE_CAST_128       IKE_VAL_CAST_CBC
#else
#define IKE_CAST_128       -1
#endif
```

```
#endif

/* Twofish algorithm constants appended to the list */
#if (IKE_INCLUDE_TWOFISH == NU_TRUE)
#define IKE_TWOFISH                IKE_VAL_TWOFISH_CBC
#else
#define IKE_TWOFISH                -1
#endif
```

6. If block length of the encryption algorithm is greater than the maximum value defined by the `IKE_MAX_ENCRYPT_BLOCK_LEN` macro, then update this macro. It is defined in `os/include/networking/ike_cfg.h`.
7. Append an entry of the new encryption algorithm to the end of the `IKE_Encryption_Algos` algorithm structure in `os/networking/ike/ike_cfg.c`. Members of this structure are described in [Table 8-19](#).

**Table 8-19. IKE\_Encryption\_Algos**

Members	Description
<code>ike_algo_identifier</code>	(UINT8) Specifies the IKE algorithm identifier defined in Step 5.
<code>crypto_algo_id</code>	(UINT8) Specifies the crypto algorithm identifier. This is the identifier assigned to the algorithm in Step 1.

For example, add an entry for Twofish encryption to this table as follows, assuming that `IKE_OPENSSL_TWOFISH` is the crypto identifier for this algorithm:

```
/* All supported IKE Encryption algorithms */
IKE_ENCRYPTION_ALGO IKE_Encryption_Algos[IKE_TOTAL_ENCRYPTION_ALGO]
= {
    #if (IKE_INCLUDE_DES == NU_TRUE)
    #ifdef OPENSSL_NO_DES
    #error DES encryption enabled in IKE but not in OpenSSL.
    #endif
    {
        IKE_OPENSSL_DES,
        IKE_DES
    },
    #endif
    #if (IKE_INCLUDE_3DES == NU_TRUE)
    #ifdef OPENSSL_NO_3DES
    #error 3DES encryption enabled in IKE but not in OpenSSL.
    #endif
    {
        IKE_OPENSSL_3DES,
        IKE_3DES
    },
    #endif
    #if (IKE_INCLUDE_BLOWFISH == NU_TRUE)
    #ifdef OPENSSL_NO_BF(
    #error Blowfish encryption enabled in IKE but not in OpenSSL.
    #endif
```

```
        {
            IKE_OPENSSL_BF,
            IKE_BLOWFISH
        },
    #endif
    #if (IKE_INCLUDE_CAST128 == NU_TRUE)
    #ifdef OPENSSL_NO_CAST
    #error CAST-128 encryption enabled in IKE but not in OpenSSL.
    #endif
    {
        IKE_OPENSSL_CAST,
        IKE_CAST_128
    },
    #endif
    #if (IKE_INCLUDE_TWOFISH == NU_TRUE)
    #ifdef OPENSSL_NO_TWOFISH
    #error Twofish encryption enabled in IKE but not in OpenSSL.
    #endif
    {
        IKE_OPENSSL_TWOFISH,
        IKE_TWOFISH
    },
    #endif
};
```

8. To use the new encryption algorithm, specify the algorithm identifier, defined in Step 5, in the IKE policy.

## Related Topics

[IKE Customizations](#)

[Adding Phase 1 Hash Algorithm](#)

## Adding Phase 1 Hash Algorithm

IKE can be customized to add new hash algorithms that are negotiated for IKE SAs in Phase 1.

## Procedure

Perform the following steps to add support for a new hash algorithm.

1. Add support for the new hash algorithm to the Nucleus SSL library used by IKE. Refer to the Security Sockets Layer (SSL) chapter for more information on adding new algorithms to this library. Also, add the corresponding hash identifier in the *ike\_crypto\_wrappers.h* file.
2. Ensure the Internet Assigned Numbers Authority (IANA) has assigned a Phase 1 hash algorithm ID to the algorithm being added. This ID would be defined by an RFC that describes support of the hash algorithm for IPsec and IKE. Add a macro for this algorithm to the “IKE DOI and other Phase 1 constants” section of *ike\_doi.h*, if it is not already present. For example, if SHA256 has been assigned an ID of 54321, then a macro is added as follows.

```
/* ISAKMP SA attribute values for Authentication Algos. */
#define IKE_VAL_MD5 1
#define IKE_VAL_SHA 2
#define IKE_VAL_TIGER 3
#define IKE_VAL_SHA256 54321
```

Note that in the previous example, if support for Tiger hashing is being added instead of SHA256, then no modification would be required because `IKE_VAL_TIGER` has already been defined in the default list of macros.

3. Define the `IKE_INCLUDE_XXX` macro in `os/include/networking/ike_cfg.h` where XXX is the algorithm name. If this macro is set to `NU_TRUE`, the algorithm is included in the build, and if set to `NU_FALSE`, the algorithm is excluded from the build. The following sample demonstrates adding the include macro for SHA256 hashing:

```
/* Hash Algorithms. They are specific to the phase 1
 * authentication in an IKE exchange. For configuration
 * of hash algorithms used in IPsec SAs, see the
 * Nucleus IPsec configuration file.
 */
#define IKE_INCLUDE_MD5 NU_TRUE /* MD5. */
#define IKE_INCLUDE_SHA1 NU_TRUE /* SHA1. */
#define IKE_INCLUDE_SHA256 NU_TRUE /* SHA256. */
```

4. Add the algorithm include macro, defined in Step 3, to the `IKE_TOTAL_HASH_ALGO` macro. The following is an example for adding the `IKE_INCLUDE_SHA256` hash algorithm identifier:

```
/* This macro calculates the total number of hash algorithms
 * included in the build.
 */
#define IKE_TOTAL_HASH_ALGO ((IKE_INCLUDE_MD5 + \
                               IKE_INCLUDE_SHA1 + \
                               IKE_INCLUDE_SHA256))
```

5. Define an IKE algorithm identifier and set its value equal to the IANA defined algorithm identifier, defined in Step 2. The identifier macro must be defined in `os/include/networking/ike_cfg.h`. For example, add the identifier for the SHA256 algorithm as follows:

```
/* MD5 hash algorithm constants */
#if (IKE_INCLUDE_MD5 == NU_TRUE)
#define IKE_MD5 IKE_VAL_MD5
#else
#define IKE_MD5 -1
#endif
/* SHA1 hash algorithm constants */
#if (IKE_INCLUDE_SHA1 == NU_TRUE)
#define IKE_SHA1 IKE_VAL_SHA
#else
#define IKE_SHA1 -1
#endif
/* SHA256 hash algorithm constants */
```



```
#if (IKE_INCLUDE_SHA256 == NU_TRUE)
#define IKE_SHA256                IKE_VAL_SHA
#else
#define IKE_SHA256                -1
#endif
```

6. If the digest length of the hash algorithm is greater than the maximum value defined by the `IKE_MAX_HASH_DATA_LEN` macro, then update this macro. It is defined in *os/include/networking/ike\_cfg.h*.
7. Append an entry of the new hash algorithm to the end of the `IKE_Hash_Algos` algorithm structure in *os/networking/ike/ike\_cfg.c*. Members of this structure are described in [Table 8-20](#)

**Table 8-20. IKE\_HASH\_Algos**

Member	Description
ike_algo_identifier	(UINT8) Specifies the IKE algorithm identifier defined in Step 5.
crypto_algo_id	(UINT8) Specifies the crypto algorithm identifier. This would be the identifier assigned to the algorithm in Step 1.

For example, add an entry for SHA256 hashing to this table as follows:

```
/* All supported IKE Hash algorithms */
IKE_HASH_ALGO IKE_Hash_Algos[IKE_TOTAL_HASH_ALGO] = {
#if (IKE_INCLUDE_MD5 == NU_TRUE)
#ifdef OPENSSL_NO_MD5
#error MD5 hash algorithm enabled in IKE but not in OpenSSL.
#endif
{
    IKE_OPENSSL_MD5,
    IKE_MD5,
},
#endif
#if (IKE_INCLUDE_SHA1 == NU_TRUE)
#ifdef OPENSSL_NO_SHA1
#error SHA1 hash algorithm enabled in IKE but not in OpenSSL.
#endif
{
    IKE_OPENSSL_SHA1,
    IKE_SHA1,
},
#endif
#if (IKE_INCLUDE_SHA256 == NU_TRUE)
#ifdef OPENSSL_NO_SHA256
#error SHA256 hash algorithm enabled in IKE but not in OpenSSL.
#endif
{
    IKE_OPENSSL_SHA256,
    IKE_SHA256,
},
#endif
}
```

8. To use the new hash algorithm specify the algorithm identifier, defined in Step 5, in the IKE policy.

## Related Topics

[IKE Customizations](#)

[Adding Phase 1 Encryption Algorithm](#)

## Adding Phase 2 Encryption Algorithm

IKE can be customized to add new encryption algorithms that are negotiated for IPsec SAs in Phase 2.

### Procedure

Perform the following steps to add support for a new encryption algorithm.

1. Add support for the new encryption algorithm to the Nucleus SSL library. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for more information on adding new algorithms to this product
2. Add support for the new encryption algorithm to Nucleus IPsec. Refer to the [IPsec](#) chapter for more information on adding new algorithms to this product.
3. Ensure the Internet Assigned Numbers Authority (IANA) has assigned a Phase 2 encryption algorithm ID to the algorithm being added. This ID would be defined by an RFC that describes support of the encryption algorithm for IPsec and IKE. Add a macro for this algorithm to the “IPsec DOI constants” section of `ike_doi.h`, if it is not already present. For example, if Twofish has been assigned an ID of 12345, then the Twofish macro is added as follows:

```
/* Transform IDs for ESP. */
#define IKE_IPS_TRANS_ESP_DES_IV64      1
#define IKE_IPS_TRANS_ESP_DES          2
#define IKE_IPS_TRANS_ESP_3DES         3
#define IKE_IPS_TRANS_ESP_RC5          4
#define IKE_IPS_TRANS_ESP_IDEA         5
#define IKE_IPS_TRANS_ESP_CAST         6
#define IKE_IPS_TRANS_ESP_BLOWFISH     7
#define IKE_IPS_TRANS_ESP_3IDEA        8
#define IKE_IPS_TRANS_ESP_DES_IV32     9
#define IKE_IPS_TRANS_ESP_RC4          10
#define IKE_IPS_TRANS_ESP_NULL         11
#define IKE_IPS_TRANS_ESP_AES          12
#define IKE_IPS_TRANS_ESP_TWOFISH     12345
```

Note that in the previous example, if support for IDEA encryption is being added instead of Twofish, then no modification would be required because `IKE_IPS_TRANS_ESP_IDEA` has already been defined in the default list of macros.

4. Add support for mapping the Nucleus IPsec identifier for the encryption algorithm (defined during Step 2) to the IANA assigned algorithm identifier (defined in Step 3).

This would require addition of a switch case to the `IKE_ESP_Trans_ID_IPS_To_IKE` function defined in `src/ike_ips.c`. For example, add support for Twofish encryption to this function as follows:

```
UINT8 IKE_ESP_Trans_ID_IPS_To_IKE(UINT8 ips_encrypt_algo)
{
    UINT8          transform_id;

    /* Determine the IPsec encryption algorithm ID. */
    switch(ips_encrypt_algo)
    {
        ...

        /* Start of code addition for Twofish */
#ifdef (IPSEC_INCLUDE_TWOFISH == NU_TRUE)
        /* Assuming that the IPsec algorithm identifier for
         * Twofish was named IPSEC_TWOFISH_CBC.
         */
        case IPSEC_TWOFISH_CBC:
            /* Set the IKE transform ID for Twofish. This was
             * defined in ike_doi.h during Step 3.
             */
            transform_id = IKE_IPS_TRANS_ESP_TWOFISH;
            break;
#endif
        /* End of code addition for Twofish */

        default:
            /* Unrecognized encryption algorithm ID */
            transform_id = 0;
            break;
    }

    /* Return the transform ID. */
    return (transform_id);
} /* IKE_ESP_Trans_ID_IPS_To_IKE */
```

5. Add support for mapping the IANA assigned algorithm identifier (defined in Step 3) to the Nucleus IPsec identifier for the encryption algorithm (defined during Step 2). This requires the addition of a switch case to the `IKE_ESP_Trans_ID_IKE_To_IPS` function defined in `src/ike_ips.c`. For example, add support for Twofish encryption to this function as follows:

```
UINT8 IKE_ESP_Trans_ID_IKE_To_IPS(UINT8 transform_id)
{
    UINT8          ips_encrypt_algo;

    /* Determine the IKE transform ID. */
    switch(transform_id)
    {
        ...

        /* Start of code addition for Twofish */
#ifdef (IPSEC_INCLUDE_TWOFISH == NU_TRUE)
```

```
/* Below is the IKE transform ID for Twofish. This
 * was defined in ike_doi.h during Step 3.
 */
case IKE_IPS_TRANS_ESP_TWOFISH:
    /* Assuming that the IPsec algorithm identifier
     * for Twofish was named IPSEC_TWOFISH_CBC
     */
    ips_encrypt_algo = IPSEC_TWOFISH_CBC;
    break;
#endif
/* End of code addition for Twofish */

default:
    /* Unrecognized IKE transform ID. */
    ips_encrypt_algo = 0;
    break;
}

/* Return the algorithm ID. */
return (ips_encrypt_algo);

} /* IKE_ESP_Trans_ID_IKE_To_IPS */
```

6. To use the new encryption algorithm, specify its IPsec algorithm identifier in the Nucleus IPsec policy. Refer to the [IPsec](#) chapter for more information on specifying encryption algorithms in the policy.

## Related Topics

[IKE Customizations](#)

[Adding Phase 2 Hash Algorithm](#)

## Adding Phase 2 Hash Algorithm

IKE can be customized to add new hash algorithms that are negotiated for IPsec SAs in Phase 2. The new hash algorithm can be configured to support both the ESP and AH protocols.

## Procedure

Perform the following steps to add support for a new hash algorithm:

1. Add support for the new hash algorithm to the Nucleus SSL library. Refer to the [Security Sockets Layer \(SSL\)](#) chapter for more information on adding new algorithms to this product.
2. Add support for the new hash algorithm to Nucleus IPsec. Refer to the [IPsec](#) chapter for more information on adding new algorithms to this product.
3. Ensure the Internet Assigned Numbers Authority (IANA) has assigned Phase 2 hash algorithm identifiers to the algorithm being added. These identifiers would be defined by an RFC that describes support of the hash algorithm for IPsec and IKE. Add macros for this algorithm to the “IPsec DOI constants” section of *ike\_doi.h*, if they are not

already present. For example, if SHA256 has been assigned an ID of 12345 for AH and an ID of 54321 for ESP, then the SHA256 macros are added as follows:

```
/* Transform IDs for AH. */
#define IKE_IPS_TRANS_AH_MD5          2
#define IKE_IPS_TRANS_AH_SHA          3
#define IKE_IPS_TRANS_AH_DES          4
#define IKE_IPS_TRANS_AH_SHA256       12345

...

/* SA attribute value for Authentication algo */
#define IKE_IPS_VAL_HMAC_MD5           1
#define IKE_IPS_VAL_HMAC_SHA           2
#define IKE_IPS_VAL_DES_MAC            3
#define IKE_IPS_VAL_KPDK               4
#define IKE_IPS_VAL_SHA256             54321
```

Note that in the previous example, if support for DES-MAC based hashing was being added instead of SHA256, then no modification would be required because `IKE_IPS_TRANS_AH_DES` and `IKE_IPS_VAL_DES_MAC` have already been defined in the default list of macros.

4. Add support for mapping the Nucleus IPsec identifier for the hash algorithm (defined during Step 2) to the IANA assigned algorithm identifiers (defined in Step 3). This requires the addition of a switch case to the `IKE_AH_Trans_ID_IPS_To_IKE` and `IKE_Auth_Algo_ID_IPS_To_IKE` functions defined in `src/ike_ips.c`. For example, add support for SHA256 hashing for AH as follows:

```
UINT8 IKE_AH_Trans_ID_IPS_To_IKE(UINT8 ips_auth_algo)
{
    UINT8          transform_id;

    /* Determine the IPsec algorithm. */
    switch(ips_auth_algo)
    {
        ...

        /* Start of code addition for SHA256 */
        #if (IPSEC_INCLUDE_SHA256 == NU_TRUE)
        /* IPsec identifier for SHA256 */
        case IPSEC_HMAC_SHA_256_96:
            /* Set the AH transform ID for SHA256. This was
             * defined in ike_doi.h during Step 3.
             */
            transform_id = IKE_IPS_TRANS_AH_SHA256;
            break;
        #endif
        /* End of code addition for SHA256 */

        default:
            /* Unrecognized IPsec algorithm ID */
            transform_id = 0;
            break;
    }
}
```

```
    }

    /* Return the transform ID. */
    return (transform_id);

} /* IKE_AH_Trans_ID_IPS_To_IKE */
```

Add support for SHA256 hashing for ESP as follows:

```
UINT16 IKE_Auth_Algo_ID_IPS_To_IKE(UINT8 ips_auth_algo)
{
    UINT16      auth_algo;

    /* Determine the algorithm type. */
    switch(ips_auth_algo)
    {
        ...

        /* Start of code addition for SHA256 */
#ifdef (IPSEC_INCLUDE_SHA256 == NU_TRUE)
        /* IPsec identifier for SHA256 */
        case IPSEC_HMAC_SHA_256_96:
            /* Set the ESP attribute ID for SHA256. This was
             * defined in ike_doi.h during Step 3.
             */
            auth_algo = IKE_IPS_VAL_HMAC_SHA256;
            break;
#endif

        /* End of code addition for SHA256 */

        default:
            /* Unrecognized IPsec algorithm ID */
            auth_algo = 0;
            break;
    }

    /* Return the IKE algorithm ID. */
    return (auth_algo);

} /* IKE_Auth_Algo_ID_IPS_To_IKE */
```

5. Add support for mapping the IANA assigned algorithm identifiers (defined in Step 3) to the Nucleus IPsec identifier for the hash algorithm (defined during Step 2). This requires the addition of a switch case to the `IKE_AH_Trans_ID_IKE_To_IPS` and `IKE_Auth_Algo_ID_IKE_To_IPS` functions defined in `src/ike_ips.c`. For example, add support for SHA256 hashing for AH as follows:

```
UINT8 IKE_AH_Trans_ID_IKE_To_IPS(UINT8 transform_id)
{
    UINT8      ips_auth_algo;
    /* Determine the transform type. */
    switch(transform_id)
    {
        ...

        /* Start of code addition for SHA256 */
```

```

#if (IPSEC_INCLUDE_SHA256 == NU_TRUE)
/* AH transform ID for SHA256. This was
 * defined in ike_doi.h during Step 3.
 */
case IKE_IPS_TRANS_AH_SHA256:
/* IPsec identifier for SHA256 */
ips_auth_algo = IPSEC_HMAC_SHA_256_96;
break;
#endif
/* End of code addition for SHA256 */
default:
/* Unrecognized transform ID */
ips_auth_algo = 0;
break;
}
/* Return the algorithm ID. */
return (ips_auth_algo);
} /* IKE_AH_Trans_ID_IKE_To_IPS */

Add support for SHA256 hashing for ESP as follows:
UINT8 IKE_Auth_Algo_ID_IKE_To_IPS(UINT16 auth_algo)
{
    UINT8            ips_auth_algo;

    /* Determine the algorithm type. */
    switch(auth_algo)
    {
        ...

        /* Start of code addition for SHA256 */
#if (IPSEC_INCLUDE_SHA256 == NU_TRUE)
/* ESP attribute ID for SHA256. This was
 * defined in ike_doi.h during Step 3.
 */
case IKE_IPS_VAL_HMAC_SHA256:
/* Set the IPsec identifier for SHA256. */
ips_auth_algo = IPSEC_HMAC_SHA_256_96;
break;
#endif
/* End of code addition for SHA256 */

default:
/* Unrecognized IKE algorithm ID */
ips_auth_algo = 0;
break;
}

/* Return the IPsec algorithm ID. */
return (ips_auth_algo);

} /* IKE_Auth_Algo_ID_IKE_To_IPS */

```

6. To use the new hash algorithm, specify its IPsec algorithm identifier in the Nucleus IPsec policy. Refer to the [IPsec](#) chapter for more information on specifying hash algorithms in the policy.

## Related Topics

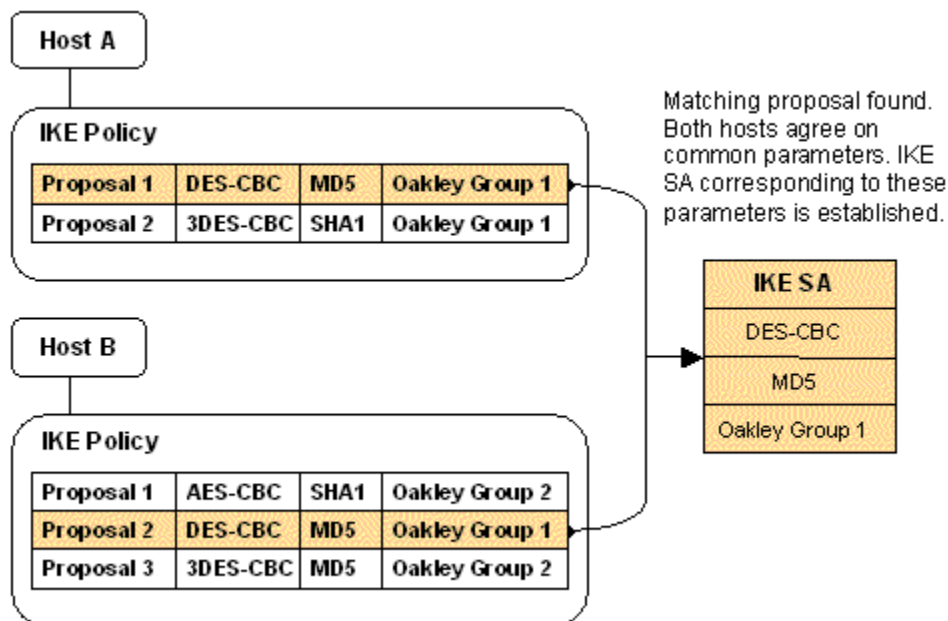
[IKE Customizations](#)

[Adding Phase 2 Encryption Algorithm](#)

# IKE Proposals

Both Phase 1 and Phase 2 of an IKE exchange involve the proposal and selection of SA parameters. Proposals advertised in Phase 1 are derived from the IKE policy being used for the exchange. Each item in the `ike_xchg1_attris` member of the IKE policy results in an independent proposal. The total number of proposals is equal to the number of items in the attributes array. An example of a Phase 1 negotiation is shown in [Figure 8-6](#). Some proposal parameters have been omitted for simplicity.

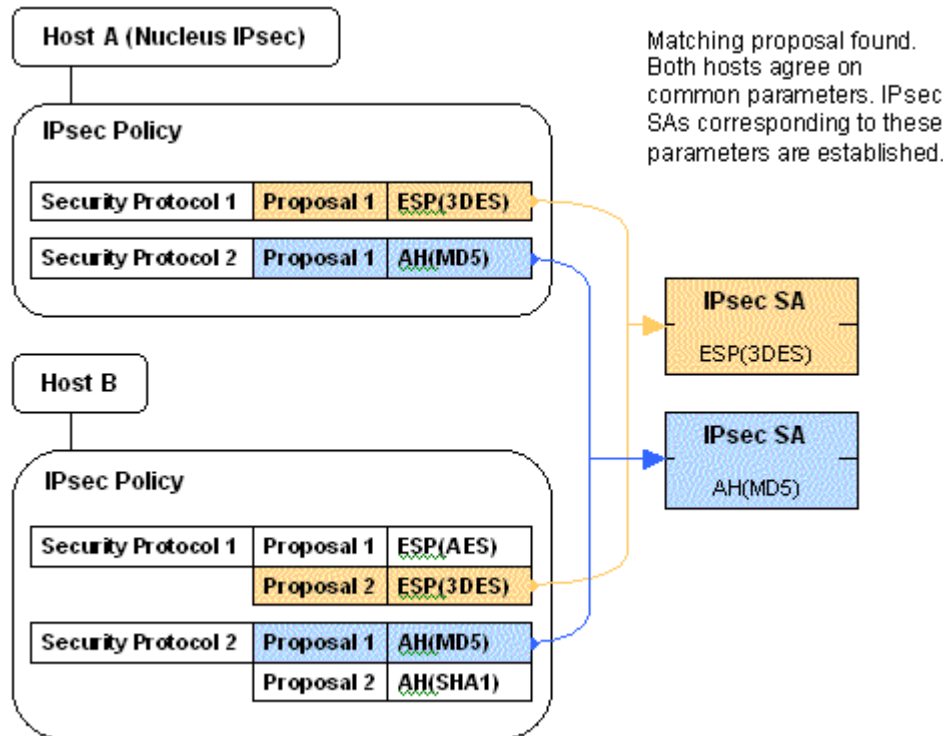
**Figure 8-6. Phase One Negotiation**





Similarly, proposals advertised in Phase 2 are derived from the IPsec policy being used for the exchange. One major difference between Phase 1 and Phase 2 proposals is that only a single proposal can be specified by the IPsec policy. This is a constraint of Nucleus IPsec, not of the IPsec protocol. An example of a Phase 2 negotiation is shown in the following diagram. Host A is running Nucleus IPsec. This is why there is only a single proposal for each security protocol in the IPsec policy. Host B is running a third party IPsec implementation, which supports multiple proposals.

**Figure 8-7. Phase Two Negotiation**



## Exchange Invocation

An IKE exchange is normally invoked when required SAs are missing and need to be established to allow communication. It could also be requested explicitly by an application. [Table 8-21](#) lists different situations that lead to an IKE exchange.

**Table 8-21. IKE Exchange**

<b>Situation</b>	<b>Description</b>
Exchange requested by a remote host.	This is the case when a remote host initiates an IKE exchange. The local host responds to the exchange if the IKE service is running.
Exchange requested by Nucleus IPsec due to missing SAs.	When sending a packet, one or more SAs might be missing. An exchange is initiated to establish the missing SAs. This whole process is not visible to the upper-layer applications and is carried out transparently.
Exchange requested by Nucleus IPsec on soft lifetime expiry of a SA.	IKE sets a soft lifetime of SAs when they are established. The soft lifetime expires a few seconds before the hard lifetime. As a result, an IKE exchange is initiated before the SAs are removed from the database. This eliminates the delay in communication that normally occurs when IPsec must wait for an IKE exchange to complete.
Explicit exchange request by a user application.	Higher-layer applications are normally unaware of IKE exchanges. This is the only case in which an application may explicitly initiate an IKE exchange using the IKE_Initiate API function. An explicit initiation ensures that all required SAs are in place before the actual communication begins. However, an explicit invocation is not required in most cases.

In the first three situations, IKE exchanges are handled internally by Nucleus IPsec, provided that valid IKE and IPsec policies are in place. However, in the fourth situation, the requesting application must be aware of IPsec policies and must accordingly request an exchange.

## IPsec SA Re-keying

When an IPsec SA expires, the time taken for a new SA negotiation introduces a delay in the packet stream. Packets which require security cannot be sent until IKE re-establishes the IPsec SA. This undesirable behavior can be avoided by the use of SA re-keying on soft lifetime expiry.

IPsec SAs have a certain lifetime after which they must be deleted. This is called a hard lifetime. Similarly, a soft lifetime could also be assigned to a SA. Expiry of the soft lifetime does not result in deletion of the SA, but instead it is considered a warning that an SA is about to expire. As a result, an IKE negotiation is initiated prior to the actual expiry of the IPsec SA.

Automatic SA re-keying can be used in Nucleus IPsec by specifying the `IPSEC_REFRESH_SA` action in the IPsec policy. This action can only be set on the SA's soft lifetime.

To limit resources utilized by SAs, Nucleus IPsec enforces two conditions for SA re-keying. Both of the following conditions must be satisfied for an SA to be re-keyed:

- SA has been used at least once during its lifetime. Either by inbound or outbound packets.
- SA has not been explicitly deleted by the remote host.

For more information about the `IPSEC_REFRESH_SA` flag, see the [IPSEC\\_SA\\_LIFETIME](#) section of the [IPsec](#) chapter.

## SA Re-keying with Microsoft Windows

Microsoft Windows has a strict policy for idle SAs. If an IPsec SA is idle for a few minutes, Windows deletes the SA before its actual lifetime expiry. The remote host is notified of this SA deletion by an informational message. This behavior may cause the previously mentioned second condition to fail. Therefore in this case, an IPsec SA is not re-keyed even if the `IPSEC_REFRESH_KEY` action is specified.

## IKE Error Codes

The following macros and their associated values are error codes returned by IKE API functions are shown in [Table 8-22](#) :

**Table 8-22. IKE Error Codes**

Code	Value	Meaning
<code>IKE_LENGTH_IS_SHORT</code>	-4760	Input or output packet buffer not large enough.
<code>IKE_ALREADY_EXISTS</code>	-4761	Item being added to the database already exists.
<code>IKE_NO_MEMORY</code>	-4762	Not enough memory to carry out exchange.
<code>IKE_NO_UPDATE</code>	-4763	Indicates that exchange status has not changed since the last update.
<code>IKE_INVALID_LENGTH</code>	-4764	Specified length of a parameter is invalid.
<code>IKE_INVALID_STATE</code>	-4765	Packet received does not match current state of the state machine.

**Table 8-22. IKE Error Codes (cont.)**

Code	Value	Meaning
IKE_INVALID_DOMAIN	-4766	Domain name specified by remote host is invalid.
IKE_INVALID_SELECTOR	-4767	IPsec selector specified by remote host is invalid.
IKE_UNSUPPORTED_METHOD	-4768	Unsupported authentication method requested.
IKE_UNSUPPORTED_IDTYPE	-4769	Unsupported identifier type encountered.
IKE_UNSUPPORTED_NOTIFY	-4770	Unsupported notification message received.
IKE_UNALLOWED_MODE	-4771	Requested exchange mode not allowed by policy.
IKE_SA2_NOT_FOUND	-4772	Nucleus IPsec policy does not define any security protocols to be negotiated.
IKE_UNEXPECTED_MESSAGE	-4773	Unexpected IKE message received.
IKE_TRANSFORM_MISMATCH	-4774	IKE negotiation failed due to transform mismatch.
IKE_INDEX_NOT_FOUND	-4775	Too many exchanges in progress already. The maximum limit is defined by the <code>IKE_MAX_WAIT_EVENTS</code> configuration macro.
IKE_NOT_BUFFERED	-4776	Message resend failed because an IKE buffer is not available.
IKE_IS_RESEND	-4777	Received and ignored a resent IKE message.
IKE_NO_KEYMAT	-4778	IKE exchange has not negotiated any keys yet.
IKE_INTERNAL_ERROR	-4779	An internal system error occurred.
IKE_LENGTH_IS_LONG	-4780	Specified length of a parameter is too long.
IKE_PHASE1_INCOMPLETE	-4781	Phase 1 of exchange is currently in progress.
IKE_PHASE2_INCOMPLETE	-4782	Phase 2 of exchange is currently in progress.
IKE_PHASE1_TIMED_OUT	-4783	Phase 1 of exchange timed out and was aborted.
IKE_PHASE2_TIMED_OUT	-4784	Phase 2 of exchange timed out and was aborted.

**Table 8-22. IKE Error Codes (cont.)**

Code	Value	Meaning
IKE_SA_TIMED_OUT	-4785	IKE SA timed out during exchange.
IKE_ADDR_MISMATCH	-4786	Received IKE message from unexpected host.
IKE_UNALLOWED_XCHG	-4787	Phase 1 exchange not allowed by IKE policy.
IKE_UNALLOWED_XCHG2	-4788	Phase 2 exchange not allowed by Nucleus IPsec policy.
IKE_UNEQUAL_PLOAD_LEN	-4900	Invalid payload format in incoming message.
IKE_INVALID_COOKIE	-4901	Cookie in incoming message is invalid.
IKE_INVALID_MJR_VER	-4902	IKE version mismatch between the two peers.
IKE_INVALID_MNR_VER	-4903	IKE version mismatch between the two peers.
IKE_INVALID_PLOAD_TYPE	-4904	Unrecognized payload type received.
IKE_INVALID_XCHG_TYPE	-4905	Invalid exchange type requested.
IKE_INVALID_FLAGS	-4906	Flags in received message are invalid.
IKE_INVALID_MSGID	-4907	Message ID of received message is invalid.
IKE_INVALID_PAYLOAD	-4908	Payload in received message is invalid.
IKE_DUPLICATE_PAYLOAD	-4909	Duplicate payloads received.
IKE_TOO_MANY_TRANSFORMS	-4910	Too many transforms in the proposal. The maximum limit is defined by the <code>IKE_MAX_TRANSFORMS</code> configuration macro.
IKE_TOO_MANY_PROPOSALS	-4911	Too many proposal payloads in the proposal. The maximum limit is defined by the <code>IKE_MAX_PROPOSALS</code> configuration macro.
IKE_SA_NOT_FOUND	-4912	A required IKE SA was not found.
IKE_ATTRIB_TOO_LONG	-4913	Variable length attribute is too long.
IKE_INVALID_SPI	-4914	SPI proposed by remote host is invalid.
IKE_INVALID_PROTOCOL	-4915	Unrecognized protocol in exchange.

**Table 8-22. IKE Error Codes (cont.)**

Code	Value	Meaning
IKE_INVALID_TRANSFORM	-4916	Invalid transform in received proposal.
IKE_INVALID_KEYLEN	-4917	Unsupported algorithm key length requested.
IKE_INVALID_PROPOSAL	-4918	Invalid proposal format.
IKE_UNSUPPORTED_DOI	-4919	Requested Domain of Interpretation (DOI) is not supported.
IKE_UNSUPPORTED_ALGO	-4920	Requested algorithm is not supported.
IKE_UNSUPPORTED_ATTRIB	-4921	Unrecognized attribute in received proposal.
IKE_MISSING_ATTRIB	-4922	A required attribute is missing from the proposal.
IKE_MISSING_PAYLOAD	-4923	A required payload is missing from the message.
IKE_UNEXPECTED_PAYLOAD	-4924	Unexpected payload received in the message.
IKE_NOT_NEGOTIABLE	-4925	Phase 1 or Phase 2 proposal is unacceptable.
IKE_PROPOSAL_TAMPERED	-4926	Proposal originally proposed has been tampered.
IKE_AUTH_FAILED	-4927	Authentication failed.
IKE_VERIFY_FAILED	-4928	Message hash verification failed.
IKE_ID_MISMATCH	-4929	Remote host's identification mismatch.
IKE_UNSUPPORTED_SITU	-4930	Unsupported situation field in IKE message.
IKE_INVALID_ID	-4931	Invalid identifier received.
IKE_GEN_ERROR	-4932	Error occurred in some other library that IKE is using.
IKE_CERT_FILE_ERROR	-4933	Certificate file could not be read.
IKE_CERT_ERROR	-4934	Certificate file read but certificate could not be parsed.
IKE_INVALID_PARAMS	-5003	One or more parameters are invalid.
IKE_NOT_FOUND	-5001	No matching IKE group or policy found.









































































































































# Chapter 9

## Simple Network Management Protocol (SNMP)

---

### Warning



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## SNMP Overview

SNMP defines a framework for the management of network-capable devices. It is described by a set of core standards outlined in a series of Request for Comments (RFCs). RFCs set forth standards that define the structure, organization, and implementation of Internet protocols. SNMP uses a manager/agent model and operates at the transport layer as an application protocol. SNMP is the overwhelming choice of network architects because it lives up to its name - simple. It is supported by most applications and is well known and understood.

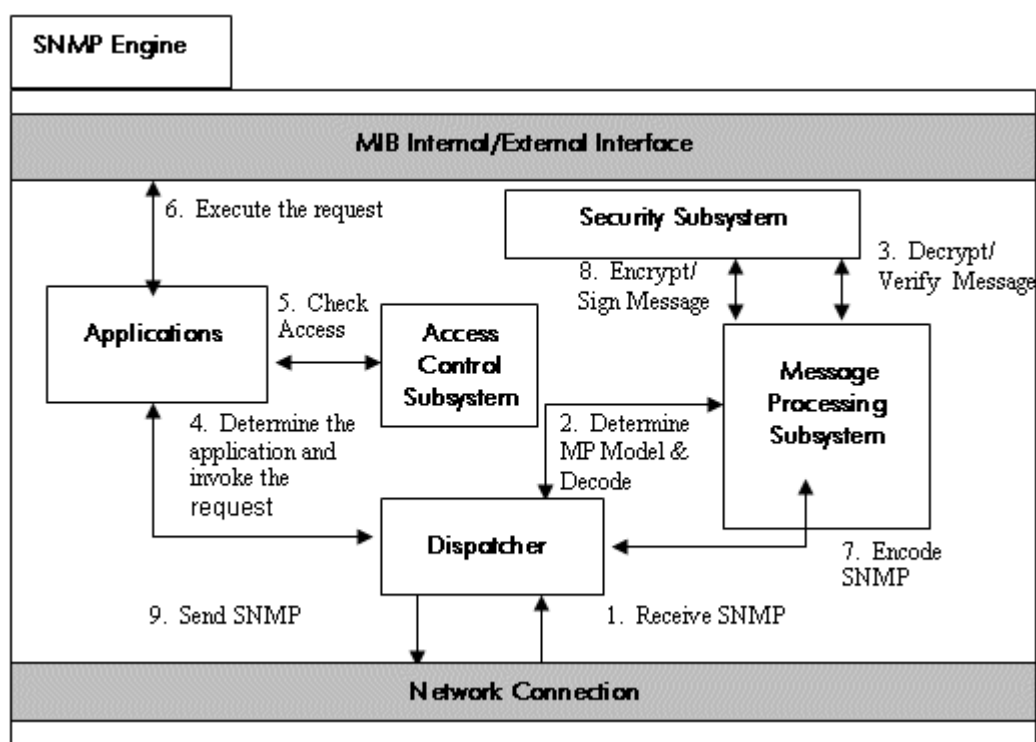
Nucleus SNMP is an embedded implementation of the management protocol and is provided as two separate products. One product, Nucleus SNMP, supplies only version 1 functionality of the protocol. The other package, Nucleus SNMPv3, provides versions 1, 2c, and 3 agents. SNMPv3 enables processing of multiple message formats with multiple security models. User-based Security Model (USM) and Community-Based Security Model (CBSM) are supported. User-based Security Model supports HMAC-MD5 and HMAC-SHA as Authentication Protocol and CBC-DES as Privacy Protocol. The SNMP version 3 Framework also defines a set of MIBs (Management Information Base). The parameters, which form the information base, can be configured remotely, in effect altering the behavior of the Nucleus SNMP agent.

Nucleus SNMP is tuned for applications where memory and CPU resources are limited. It is extremely portable. Nucleus SNMP is very modular, enabling easy customization.

Nucleus SNMP comes with MIB-II (RFC1213 and RFC2863) and Nucleus SNMPv3 MIB (RFC3411, RFC3412, RFC3413, RFC3414, RFC3415, and RFC 2576). Other MIB Engines are also available. Enterprise-specific MIBs can be created using the Nucleus MIB Utility.

Nucleus SNMP is highly modular, enabling addition of new models. Nucleus SNMP is divided into six subsystems as shown in [Figure 9-1](#).

Figure 9-1. SNMP Engine



## Dispatcher

The dispatcher is responsible for sending and receiving messages from the network. It dispatches messages to the Message Processing Subsystem for decoding and encoding. It also communicates with the Applications Subsystem for the actual processing of the message.

## Message Processing Subsystem

The Message Processing Subsystem allows concurrent existence of multiple Message Processing Models. Nucleus SNMP contains Nucleus SNMP Version 1 Message Processing Models. Nucleus SNMPv3 contains Nucleus SNMP Version 1, Version 2c and Version 3 Message Processing Models. Any model which implements the interface defined by the Message Processing Subsystem can be easily incorporated into Nucleus SNMP.

## Security Subsystem

The Security Subsystem implements message level security through its multiple Security Models. Nucleus SNMP implements the Community-Based Security Model Version 1. Nucleus SNMPv3 implements the Community-Based Security Model Version 1 and Version 2, as well as the User-Based Security Model. Any model which implements the interface defined by the

Security Subsystem can be easily incorporated in to Nucleus SNMP. Refer to the [“Security Subsystem Configuration”](#) on page 1693 for more information on the Security Subsystem.

## Access Control Subsystem

The Access Control Subsystem controls access to the MIBs. Nucleus SNMP implements the View-based Access Control Model. Refer to [“Access Control Subsystem Configuration”](#) on page 1708 for more information on the Access Control Subsystem.

## Applications Subsystem

The Applications Subsystem implements the Command Responder Application, responsible for processing requests from the manager and the Notification Originator which is responsible for sending notifications to the selected group of managers on the occurrence of important events.

Refer to [“Application Subsystem”](#) on page 1705 for information on processing of requests and sending of notifications.

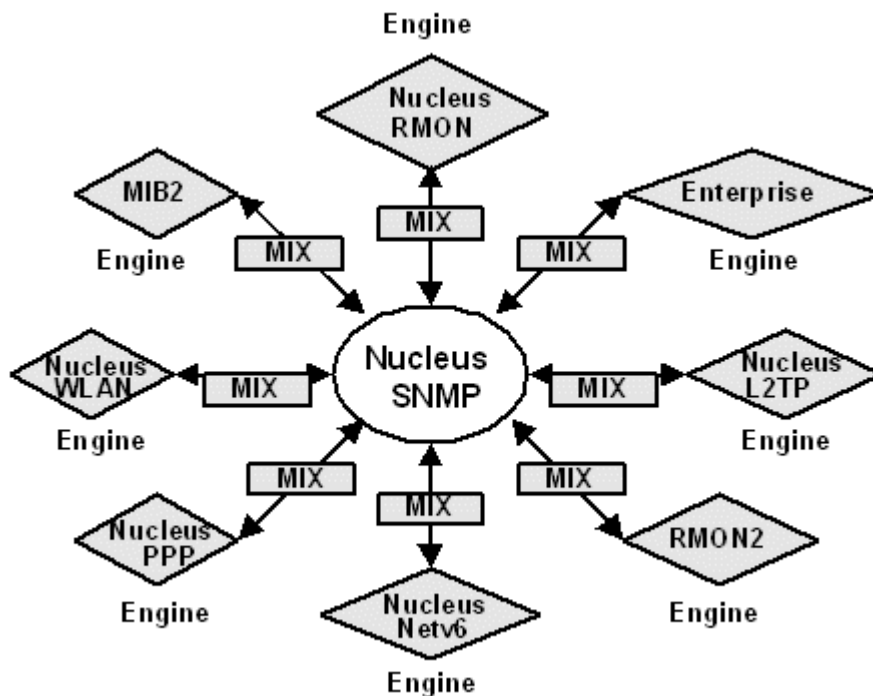
## Management Information Base (MIB) Engines

Nucleus SNMP was designed from the start to support multiple interfaces and multiple MIBs. Nucleus SNMP implements an agent core, which supports external MIB Engines. A MIB Engine is the code, which implements the MIB objects, and the support functions which enable collection of information in support of these objects. The MIB Engines are satellites bound by a well-defined interface, the MIB Internal/eXternal interface (MIX) to the central agent. The MIB Engines and agent share a single memory space.

The MIB Engine and agent interfaces are expanded such that the engines may be distributed. [Figure 9-2](#) illustrates these concepts. Additional information can be found in the following sections:

- [Nucleus MIB-II Defines](#)
- [MIB-II: Managed Objects](#)
- [Nucleus SNMPv3 MIB](#)
- [Nucleus SNMPv3 MIB: Defines](#)
- [Nucleus SNMPv3 MIB Managed Objects](#)
- [Nucleus MIB Utility](#)
- [PPP MIB](#)

Figure 9-2. Nucleus SNMP/MIB Engines



The MIB objects are installed into an internal, fast database through a registration and initialization procedure.

## Configuring SNMP Defines

This section provides SNMP parameters in the form of initialization strings, defined constants or table sizes. These parameters allow users to customize their configuration.

### `include/networking/snmp_prt.h`

The user-definable region specific to the available MIB is noted in the `include/networking/snmp_prt.h` file. Although you may define other variables, normally only those within this definable region are required. The following constant definitions are used to configure the available MIB:

- `MIBII_SYSCONTACT`
- `MIBII_SYSDESCRIPTION`
- `MIBII_SYSLOCATION`
- `MIBII_OBJECTID`
- `MIBII_SERVICES`

## MIBII\_SYSCONTACT

MIBII\_SYSCONTACT sets the sysContact string in the RFC1213 and RFC2863 [System Group](#). This is a textual identification of the contact person for this managed node, together with information on how to contact this person. The maximum length of this string is 255.

## MIBII\_SYSDescription

MIBII\_SYSDescription sets the sysDescr string in the RFC1213 and RFC2863 [System Group](#). This is a textual description of the entity. This value should include the full name and version identification of the system's hardware type, software operating-system, and networking software. This string can only contain printable ASCII characters and is restricted to a 255-character maximum string length.

## MIBII\_SYSLOCATION

MIBII\_SYSLOCATION sets the sysLocation string in the RFC1213 and RFC2863 [System Group](#). This is the physical location of this node, for example: 'telephone closet, 3rd floor'. This string is restricted to a 255-character maximum string length.

## MIBII\_OBJECTID

MIBII\_OBJECTID sets the sysObjectID value in the RFC1213 and RFC2863 [System Group](#). This is the vendor's authoritative identification of the network management subsystem contained in the entity. This value is allocated within the SMI enterprises subtree (1.3.6.1.4.1) and provides an easy and unambiguous means for determining 'what kind of box' is being managed. For example, if vendor 'Flintstones, Inc.' was assigned the subtree 1.3.6.1.4.1.4242, it could assign the identifier 1.3.6.1.4.1.4242.1.1 to its 'Fred Router'.

## MIBII\_SERVICES

MIBII\_SERVICES sets the sysServices value in the RFC1213 and RFC2863 [System Group](#). sysServices is a value indicating the set of services offered by this entity. The value is a sum. This sum initially takes the value zero, then for each layer, L, in the range 1 through 7 for which this node performs transactions,  $2^{L-1}$  is added to the sum. For example, a node which performs primarily routing functions would have a value of 4 ( $2^{3-1}$ ). In contrast, a node which is a host offering application services would have a value of 72 ( $2^{4-1} + 2^{7-1}$ ). [Table 9-1](#) provides the values for each service to be provided.

**Table 9-1. MIBII\_SERVICES**

Layer (L)	Functionality (Service)
1	Physical (for example, repeaters)
2	Data link/subnetwork (for example, bridges)
3	Internet (for example, IP gateways)

**Table 9-1. MIBII\_SERVICES (cont.)**

Layer (L)	Functionality (Service)
4	End-to-end (for example, IP hosts)
7	Applications (for example, mail relays)

For systems including OSI protocols, layers 5 and 6 may also be counted.

## include/networking/snmp.h

The following are some important defines found in this file.

- [SNMP\\_PORT](#)
- [SNMP\\_TRAP\\_PORT](#)

### SNMP\_PORT

SNMP\_PORT sets the agent port. The Nucleus SNMP manager uses the agent port to contact the agent. This port should only be changed for security reasons, and then only if necessary since any other port, other than the defined one, makes the agent non-compliant and not interoperable with other managers.

### SNMP\_TRAP\_PORT

SNMP\_TRAP\_PORT sets the managers port. The Nucleus SNMP agent uses the managers port to contact the manager. This port should only be changed for security reasons, and then only if necessary since any other port, other than the defined one, makes the agent non-compliant and not interoperable with other managers.

## snmp\_cfg.h

The following are some important defines found in this file:

- [SNMP\\_V1\\_BUFFER\\_SIZE](#)
- [SNMP\\_V2\\_BUFFER\\_SIZE](#)
- [SNMP\\_V3\\_BUFFER\\_SIZE](#)
- [SNMP\\_CR\\_QSIZE](#)
- [SNMP\\_ENABLE\\_FILE\\_STORAGE](#)
- [SNMP\\_CONFIG\\_SERIAL](#)
- [SNMP\\_DRIVE](#)
- [SNMP\\_DIR](#)



- [SNMP\\_FILE](#)
- [USM\\_FILE](#)
- [CBSM\\_FILE](#)
- [VACM\\_SEC2GROUP\\_STRUCT\\_FILE](#)
- [VACM\\_ACCESS\\_STRUCT\\_FILE](#)
- [VACM\\_VIEWTREE\\_STRUCT\\_FILE](#)
- [SNMP\\_IANA\\_NUMBER](#)

## **SNMP\_V1\_BUFFER\_SIZE**

SNMP\_V1\_BUFFER\_SIZE the message buffer size for Message Processing Model Version 1. This should be at least 150.

## **SNMP\_V2\_BUFFER\_SIZE**

SNMP\_V2\_BUFFER\_SIZE the message buffer size for Message Processing Model Version 2c. This should be at least 150.

## **SNMP\_V3\_BUFFER\_SIZE**

SNMP\_V3\_BUFFER\_SIZE the message buffer size for Message Processing Model Version 3. This should be at least 300.

## **SNMP\_CR\_QSIZE**

SNMP\_CR\_QSIZE the queue size for the Command Responder Application. The dispatcher adds all decoded requests to this queue for the command responder to respond to. This macro also defines the maximum number requests under processing at a time.

## **SNMP\_ENABLE\_FILE\_STORAGE**

Set SNMP\_ENABLE\_FILE\_STORAGE to NU\_TRUE to enable storage of MIB Data to file system. VACM, USM and CBSM MIB stores data to the files when this is enabled.

## **SNMP\_CONFIG\_SERIAL**

When running for the first time, Nucleus SNMP can be manually configured through the serial interface. Set SNMP\_CONFIG\_SERIAL to NU\_TRUE for configuration through the serial interface. By default a hard-coded configuration is used. If a real file system is used, Nucleus SNMP initializes itself using the file containing the most recent configuration.

## SNMP\_DRIVE

When using a file system, Nucleus SNMP sets SNMP\_DRIVE to its working drive.

## SNMP\_DIR

When using a file system, Nucleus SNMP sets SNMP\_DIR to its working directory.

## SNMP\_FILE

When using a file system, Nucleus SNMP saves data pertaining to its SNMP Engine MIB in this file. This file holds the count of Engine boots and the Engine ID. The first four bytes hold the count of Engine boots; the next four bytes hold the length of the Engine ID; the remaining bytes hold the Engine ID. The Length of Engine ID has a 32-byte maximum size; therefore, the maximum size of the file is  $8+32 = 40$  bytes.

## USM\_FILE

When using a file system, non-volatile data in the USM MIB is saved in this file. The USM\_FILE file contains entries of the USM User table. Each entry in the USM User table acquires 372 bytes if the USM MIB is excluded, and acquires 376 bytes if the USM MIB is included in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on the USM User table.

## CBSM\_FILE

When using a file system, non-volatile data in the CBSM MIB is saved in this file. The CBSM\_FILE file contains entries of the Community table. Each entry acquires 208 bytes if the CBSM MIB is excluded, and acquires 212 bytes if the CBSM MIB is included in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 of this guide for more information on CBSM.

## VACM\_SEC2GROUP\_STRUCT\_FILE

When using a file system, non-volatile data in the Security to Group table of VACM is saved in this file. Each entry in the Security to Group table acquires 76 bytes if the VACM MIB is excluded, and acquires 80 bytes when the VACM MIB is included in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 of this guide for more information on the Security to Group table.

## VACM\_ACCESS\_STRUCT\_FILE

When using a file system, non-volatile data in the VACM Access table is saved in this file. Each entry in the VACM Access table acquires 176 bytes if the VACM MIB is excluded, and 180 bytes if the VACM MIB is included in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 of this guide for more information on the VACM Access table.

## VACM\_VIEWTREE\_STRUCT\_FILE

When using a file system, non-volatile data in the View Table of VACM is saved in this file. Each entry in the View table acquires 188 bytes in the file. Refer to the "[Advanced SNMP Topics](#)" on page 1693" of this guide for more information on the View table of VACM.

## SNMP\_IANA\_NUMBER

SNMP\_IANA\_NUMBER is the IANA allotted enterprise number in hexadecimal format. If Acme Networks has been assigned {enterprises 696}, the first four octets would be assigned 0x000002B8 as decimal 696 is equivalent to 0x000002B8 hexadecimal.

# Configuring NET for SNMP Support

In order for Nucleus NET to start the Nucleus SNMP agent task upon initialization, you must modify the file *include/networking/net\_cfg.h*. Change the following macro to be true:

```
#define INCLUDE_SNMP                NU_TRUE
```

Nucleus NET must also be modified to collect MIB-II statistics. Set the following macro to true:

```
#define INCLUDE_MIB2_RFC1213        NU_TRUE
```

Nucleus NET library must be rebuilt after configuration.

## Configuring Nucleus SNMP

This section provides configuration options for Nucleus SNMP. The basic configuration options are provided below, followed by configuration options for specific features.

To configure Nucleus SNMP, the following modifications to the file *networking/snmp/snmp\_cfg.c* must be made.:

- Change the `cfg_hostid` variable to match the hexadecimal value of your device's IP address. See the [NU\\_SNMP\\_Set\\_Host\\_Id](#) API function for information on setting the local IP address during run-time.
- In the data structure `Snmp_Cfg_Tgr_Addr_Tbl`, change the IP address of an existing entry to the IP address of the machine that is running the SNMP Manager Software. If more entries are required in the data structure `Snmp_Cfg_Tgr_Addr_Tbl`, then `TGR_ADDR_TBL_SIZE` in *include/networking/snmp\_cfg.h* must also be configured. Entries to this table can also be added during run-time using the API routine [SNMP\\_Add\\_Target](#).

## SNMPv1

In the file *include/networking/snmp\_cfg.h*, set the following macros:

```
#define INCLUDE_SNMPv1  NU_TRUE
#define INCLUDE_SNMPv2  NU_FALSE
#define INCLUDE_SNMPv3  NU_FALSE
```

## SNMPv3

In the file *include/networking/snmp\_cfg.h*, set the following macros:

```
#define INCLUDE_SNMPv1  NU_TRUE
#define INCLUDE_SNMPv2  NU_TRUE
#define INCLUDE_SNMPv3  NU_TRUE
```

The previous macros can be modified as desired.

In the file *include/networking/snmp\_cfg.h*, set the following macros, or use the default values, when performing SNMPv3 queries from the SNMP Manager:

```
#define USM_USER_NAME_INITIAL      "initial"
#define USM_AUTH_PASSWORD_INITIAL  "authentic"
#define USM_PRIV_PASSWORD_INITIAL  "private"
```

USM\_USER\_NAME\_INITIAL specifies the user name and security name,  
USM\_AUTH\_PASSWORD\_INITIAL specifies the authentication password and  
USM\_PRIV\_PASSWORD\_INITIAL specifies the privacy password. You will use HMAC-MD5 as the authentication protocol and CBC-DES as the privacy protocol.

## Fast Initialization

Nucleus SNMP provides a conversion function that converts a USM password string into a key which is then used for either authentication or privacy. This conversion is processor intensive and can lead to large initialization times as well as system slow-down on slow processors. Perform the following steps for faster initialization:

- Set the macro USM\_PASSWORD\_2\_KEY in *include/networking/snmp\_cfg.h* to NU\_FALSE.
- Set the macro USM\_AUTH\_PASSWORD\_INITIAL in *include/networking/snmp\_cfg.h* with the value of the desired Authentication Key.
- Set the macro USM\_PRIV\_PASSWORD\_INITIAL in *include/networking/snmp\_cfg.h* with the value of the desired Privacy Key.
- To configure other default users in [Usm\\_User\\_Table](#), set usm\_auth\_password and usm\_priv\_password fields of [Usm\\_User\\_Table](#) in *networking/snmp/snmp\_cfg.c* with the values of the desired Authentication Key and Privacy Key respectively.

Refer to “[Advanced SNMP Topics](#)” on page 1693 of this guide for more information on User-based Security Model (USM).

## SNMP Configuration through the Serial Interface

When running for the first time or when initialization fails, Nucleus SNMP must be configured. Nucleus SNMP initialization consists of two steps. In the first step it tries to configure itself from the latest configuration using the file system. If it fails to configure itself from the file system, Nucleus SNMP configures itself using information provided in the *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* files. Some of the configuration parameters can be configured at run-time using the serial interface. Set `SNMP_CONFIG_SERIAL` to `NU_TRUE` in *include/networking/snmp\_cfg.h* to enable configuration through the serial interface. The Nucleus library must be rebuilt after this change.

[Table 9-2](#) lists the parameters that can be configured through the serial interface.

**Table 9-2. SNMP\_PARITY Parameters**

Parameters	Description
SNMP Engine ID	You can set a new Nucleus SNMP Engine ID. Only expert users should modify this value.
SNMP Engine Boots	When initialization fails Nucleus SNMP Engine Boots value gets reset. This value should be set to +1 of its previous value. If being run for the first time, this should be set to 1.
USM Configuration Option	When this value is set to 1 through serial configuration Nucleus SNMP configures the USM module as specified in <i>snmp_cfg.h</i> and <i>snmp_cfg.c</i> . When the value is set to 2, Nucleus SNMP does not create USM users during configuration.
VACM Configuration Option	Refer to RFC 3415 for information on the configuration options. When this value is set to 1 through serial configuration, Nucleus SNMP configures the VACM module as specified in <i>snmp_cfg.h</i> and <i>snmp_cfg.c</i> . When the value is set to 2, Nucleus SNMP does not create entries in the VACM Security to Group table, the VACM Access table nor the VACM MIB View table.

Configuring Nucleus SNMP through serial interface is user interactive. The following figures demonstrate the configuration of Nucleus SNMP through serial interface.

**Figure 9-3. Serial Output Example One**

```
*****
Copyright (c) 2004 MGC - Nucleus SNMP - Engine Configuration Wizard
*****
1. Set SNMP Engine ID - Current Value = 8000045001C0A80096
2. Set SNMP Engine Boots - Current Value = 1
3. Quit Wizard

Enter Choice: 1

New SNMP Engine ID = 8000045001C0A80096

*****
Copyright (c) 2004 MGC - Nucleus SNMP - Engine Configuration Wizard
*****
1. Set SNMP Engine ID - Current Value = 8000045001C0A80096
2. Set SNMP Engine Boots - Current Value = 1
3. Quit Wizard

Enter Choice: 2
```

**Figure 9-4. Serial Output Example Two**

```
*****
Copyright (c) 2004 MGC - Nucleus SNMP - Engine Configuration Wizard
*****
1. Set SNMP Engine ID - Current Value = 8000045001C0A80096
2. Set SNMP Engine Boots - Current Value = 5
3. Quit Wizard

Enter Choice: 3

SNMP Engine successfully configured!
Quitting Configuration Wizard
```

**Figure 9-5. Serial Output Example Three**

```
*****
Copyright (c) 2004 MGC - Nucleus SNMP - USM Configuration Wizard
*****
1. Default configuration as specified in snmp_cfg.c
2. No access configuration.

Enter Configuration Option: 1

*****
Copyright (c) 2004 MGC - Nucleus SNMP - VACM Configuration Wizard
*****
1. Default configuration as specified in snmp_cfg.c
2. No access configuration

Enter Configuration Option: 1
VACM successfully configured!
Quitting Configuration Wizard
```

## SNMP Data Structures

This section describes data structures used by the [SNMP API Functions](#).

- [mib\\_element\\_t](#)
- [mib\\_callback\\_t](#)
- [CBSM\\_COMMUNITY\\_STRUCT](#)
- [SNMP\\_CBSM\\_COMMUNITY\\_STRUCT](#)
- [SNMP\\_NOTIFY\\_REQ\\_STRUCT](#)
- [SNMP\\_NOTIFY\\_TABLE](#)
- [SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#)
- [SNMP\\_NOTIFY\\_FILTER\\_TABLE](#)
- [SNMP\\_SS\\_STRUCT](#)
- [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#)
- [SNMP\\_TARGET\\_PARAMS\\_TABLE](#)
- [SNMP\\_USM\\_USER\\_STRUCT](#)
- [USM\\_USERS\\_STRUCT](#)
- [USM\\_PRIV\\_PROT\\_STRUCT](#)
- [USM\\_AUTH\\_PROT\\_STRUCT](#)

- [VACM\\_SEC2GROUP](#)
- [VACM\\_SEC2GROUP\\_STRUCT](#)
- [VACM\\_ACCESS](#)
- [VACM\\_ACCESS\\_STRUCT](#)
- [VACM\\_VIEWTREE](#)
- [VACM\\_VIEWTREE\\_STRUCT](#)

## mib\_element\_t

`mib_element_t` is the typedef name for the `mib_element_s` data structure.

```
typedef struct mib_element_s
{
    UINT32          Id[SNMP_SIZE_SMALLOBJECTID];
    UINT32          IdLen;
    mib\_callback\_t Rqs;
    UINT16          Type;
    UINT16          Support;
} mib_element_t;
```

The members of the structure are defined in [Table 9-3](#). Some members of this structure have been omitted as they are used internally.

**Table 9-3. `mib_element_t`**

Member	Description
Id	Object identifier of the MIB object.
Idlen	Length of object identifier.
Rqs	Pointer to callback routine that handles the SNMP request on a MIB object. Uses the <a href="#">mib_callback_t</a> data structure.
Type	Type of MIB object where the current valid values are <code>SNMP_INTEGER</code> , <code>SNMP_OCTETSTR</code> , <code>SNMP_OBJECTID</code> , <code>SNMP_IPADDR</code> , <code>SNMP_COUNTER</code> , <code>SNMP_GAUGE</code> , <code>SNMP_TIMETICKS</code> , <code>SNMP_OPAQUE</code> , <code>SNMP_DISPLAYSTR</code> , and <code>SNMP_COUNTER64</code> .
Support	Type of operation it supports. It could have bitwise OR the value of <code>MIB_READ</code> , <code>MIB_WRITE</code> , and <code>MIB_CREATE</code> .



## Related Topics

[SNMP Data Structures](#)[SNMP\\_Mib\\_Register](#)[SNMP\\_Mib\\_Unregister](#)[mib\\_callback\\_t](#)

## mib\_callback\_t

Data structure `mib_callback_t` is defined as follows:

```
typedef UINT16(*mib_callback_t)(snmp_object_t *, UINT16, VOID *);
```

## Related Topics

[SNMP Data Structures](#)[SNMP\\_Mib\\_Register](#)[SNMP\\_Mib\\_Unregister](#)[mib\\_element\\_t](#)

## CBSM\_COMMUNITY\_STRUCT

`CBSM_COMMUNITY_STRUCT` is the typedef name for the `cbsm_community_struct` data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Community Table](#) data object.

```
typedef struct cbsm_community_struct
{
    UINT32 cbsm_transport_tag_len;
    UINT8 cbsm_community_index[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_community_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_security_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_engine_id[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_context_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_transport_tag[SNMP_SIZE_SMALLOBJECTID];
    UINT8 cbsm_community_index_len;
    UINT8 cbsm_engine_id_len;
    UINT8 cbsm_storage_type;
    UINT8 cbsm_status;
} CBSM_COMMUNITY_STRUCT;
```

The members of the structure are defined in [Table 9-4](#). Some members of this structure have been omitted as they are used internally.

**Table 9-4. CBSM\_COMMUNITY\_STRUCT**

Member	Description
<code>cbsm_transport_tag_len</code>	Length of the transport tag.
<code>cbsm_community_index</code>	A string that uniquely identifies this entry.

**Table 9-4. CBSM\_COMMUNITY\_STRUCT (cont.)**

Member	Description
cbsm_community_name	Nucleus SNMP community name referring to the cbsm_community_index.
cbsm_security_name	Security name to which this community is mapped. When this community is used in a request this security name along with security model is used to select the Security to Group table entry in VACM. Refer to the API routine " <a href="#">VACM_InsertGroup</a> ".
cbsm_engine_id	Engine ID.
cbsm_context_name	Context name. When this community is used in a request an entry in the VACM Context table is selected using this context name. This Context table is also used to select the VACM Access table Entry. Refer to " <a href="#">Advanced SNMP Topics</a> " on page 1693 of this guide for more information on VACM.
cbsm_transport_tag	Transport tag (list of groups separated by spaces or tabs). This value is used to select the transport table entry when validating the SNMP manager's IP address. Set this value to NU_NULL to permit any SNMP Manager to use this community.
cbsm_community_index_len	Length of the community index string.
cbsm_engine_id_len	Engine ID length.
cbsm_storage_type	<p>Storage type. Valid values are:</p> <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER - other storage,</li><li>• SNMP_STORAGE_VOLATILE - volatile storage,</li><li>• SNMP_STORAGE_NONVOLATILE - non-volatile storage,</li><li>• SNMP_STORAGE_PERMANENT - permanent storage,</li><li>• SNMP_STORAGE_READONLY - read-only storage.</li></ul> <p>This is available only if CBSM MIB has been included in the build. The CBSM MIB can be included in the build by setting the macro INCLUDE_MIB_CBSM to NU_TRUE in <i>include/networking/snmp_cfg.h</i>. By default it is included in the build.</p>

**Table 9-4. CBSM\_COMMUNITY\_STRUCT (cont.)**

Member	Description
cbsm_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active,</li><li>• SNMP_ROW_NOTINSERVICE - not in service,</li><li>• SNMP_ROW_NOTREADY - not ready to use,</li><li>• SNMP_ROW_DESTROY - destroy.</li></ul> This is available only if CBSM MIB has been included in the build. The CBSM MIB can be included in the build by setting the macro INCLUDE_MIB_CBSM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

## Related Topics

[SNMP Data Structures](#)[CBSM\\_Add\\_Community](#)[CBSM\\_Find\\_Community\\_Index](#)[CBSM\\_Remove\\_Community](#)[CBSM\\_Save\\_Community](#)[CBSM\\_Community\\_Table](#)

## SNMP\_CBSM\_COMMUNITY\_STRUCT

SNMP\_CBSM\_COMMUNITY\_STRUCT is the typedef name for the snmp\_cbsm\_community\_struct data structure. This data structure defines the Nucleus-supported [CBSM\\_Community\\_Table](#) global array that holds the community entries to be configured during configuration mode.

```
typedef struct snmp_cbsm_community_struct
{
    CHAR    community_name[SNMP_SIZE_SMALLOBJECTID];
    CHAR    security_name[SNMP_SIZE_SMALLOBJECTID];
    CHAR    transport_tag[SNMP_SIZE_SMALLOBJECTID];
    CHAR    context_name[SNMP_SIZE_SMALLOBJECTID];

} SNMP_CBSM_COMMUNITY_STRUCT;
```

The members of the structure are defined in [Table 9-5](#).

**Table 9-5. SNMP\_CBSM\_COMMUNITY\_STRUCT**

Members	Description
community_name	Community Name. In order to use this community you are required to input this value in the SNMP manager.
security_name	Security Name. This value is passed to VACM.
transport_tag	List of group names. This value is used to select the Transport Address table entry.

**Table 9-5. SNMP\_CBSM\_COMMUNITY\_STRUCT (cont.)**

Members	Description
context_name	Context Name. This value would be passed to VACM.

## Related Topics

[SNMP Data Structures](#)

[CBSM\\_Community\\_Table](#)

## SNMP\_NOTIFY\_REQ\_STRUCT

SNMP\_NOTIFY\_REQ\_STRUCT is the typedef name for the struct snmp\_notify\_req\_struct data structure.

```
typedef struct snmp_notify_req_struct
{
    UINT32          transport_domain;
    UINT32          timeout;
    INT32           retry_count;
    UINT32          snmp_object_list_len;
    NOTIFICATION_OID OID;
    snmp_object_t   snmp_object_list[SNMP_MAX_NOTIFICATION_OBJECTS];
    UINT8           transport_address[SNMP_MAX_IP_ADDRS];
    INT8            expect_response;
} SNMP_NOTIFY_REQ_STRUCT;
```

The members of the structure are defined in [Table 9-6](#). Some members of this structure have been omitted as they are used internally.

**Table 9-6. SNMP\_NOTIFY\_REQ\_STRUCT**

Member	Description
transport_domain	The transport domain being used. Currently, this can be UDP (1) only.
timeout	Timeout value when waiting for a response. This value is not used.
retry_count	Number of retries. This value is not used.
snmp_object_list_len	Number of objects in snmp_object_list.
OID	The notification OID. notification_oid contains the OID of the notification and OID.oid_len contains the length.
snmp_object_list	The list of objects to be sent with the notification. The maximum number of objects is determined by SNMP_MAX_NOTIFICATION_OBJECTS, defined in <i>snmp_cfg.h</i> . The default value is 3.
transport_address	The address of the SNMP Manager.

**Table 9-6. SNMP\_NOTIFY\_REQ\_STRUCT (cont.)**

Member	Description
expect_response	Indicates whether a response is expected. This value is not used.

## Related Topics

[SNMP Data Structures](#)[SNMP\\_Get\\_Notification\\_Ptr](#)[SNMP\\_Notification\\_Ready](#)

## SNMP\_NOTIFY\_TABLE

SNMP\_NOTIFY\_TABLE is the typedef name for the snmp\_notify\_table data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Notify Table](#) data object.

```
typedef struct snmp_notify_table
{
    UINT32 tag_len;
    INT32  snmp_notify_type;
    UINT32 snmp_notify_storage_type;
    UINT32 snmp_notify_row_status;
    CHAR   snmp_notify_name[MAX_NOTIFY_NAME_SIZE];
    UINT8  snmp_notify_tag[SNMP_SIZE_BUFCHR];
} SNMP_NOTIFY_TABLE;
```

The members of the structure are defined in [Table 9-7](#). Some members of this structure have been omitted as they are used internally.

**Table 9-7. SNMP\_NOTIFY\_TABLE**

Member	Description
tag_len	The Tag Length.
snmp_notify_type	Indicates whether this is a confirmed or unconfirmed class PDU. Valid values are TRAP and INFORM. Currently only TRAP is supported.

**Table 9-7. SNMP\_NOTIFY\_TABLE (cont.)**

Member	Description
snmp_notify_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage,</li><li>• SNMP_STORAGE_VOLATILE – volatile storage,</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage,</li><li>• SNMP_STORAGE_PERMANENT – permanent storage,</li><li>• SNMP_STORAGE_READONLY – read-only storage.</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_notify_row_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_notify_name	The notify name that uniquely identifies this entry.
snmp_notify_tag	The notify tag, which is used to identify the Target Address group with which this entry is associated.

## Related Topics

[SNMP Data Structures](#)

[SNMP\\_Add\\_To\\_Notify\\_Tbl](#)

[SNMP\\_Find\\_Notify\\_Entry](#)

[SNMP\\_Remove\\_From\\_Notify\\_Table](#)

[SNMP Notify Table](#)

## SNMP\_NOTIFY\_FILTER\_PROFILE\_TABLE

SNMP\_NOTIFY\_FILTER\_PROFILE\_TABLE is the typedef name for the snmp\_notify\_filter\_profile\_table data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Notify Filter Profile Table](#) data object.

```
typedef struct snmp_notify_filter_profile_table
{
    INT32 snmp_notify_filter_profile_storType;
```

```
INT32 snmp_notify_filter_profile_row_status;  
CHAR  snmp_target_params_name[MAX_FILTER_PROF_NAME_SIZE];  
CHAR  snmp_notify_filter_profile_name[MAX_FILTER_PROF_NAME_SIZE];  
}SNMP_NOTIFY_FILTER_PROFILE_TABLE;
```

The members of the structure are defined in [Table 9-8](#). Some members of this structure have been omitted as they are used internally.

**Table 9-8. SNMP\_NOTIFY\_FILTER\_PROFILE\_TABLE**

Member	Description
snmp_notify_filter_profile_storType	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_notify_filter_profile_row_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_target_params_name	Params name uniquely identifies an entry in this table. This variable is also used to map a Params entry to the Notify Filter table corresponding to the filter name in this entry.
snmp_notify_filter_profile_name	Notify Filter Profile name.

## Related Topics

[SNMP Data Structures](#)

[SNMP\\_Add\\_To\\_Profile\\_Tbl](#)

[SNMP\\_Find\\_Profile\\_Entry](#)

[SNMP\\_Remove\\_From\\_Profile\\_Table](#)

## SNMP\_NOTIFY\_FILTER\_TABLE

SNMP\_NOTIFY\_FILTER\_TABLE is the typedef name for the `snmp_notify_filter_table` data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Notify Filter Table](#) data object.

```
typedef struct snmp_notify_filter_table
{
    UINT32 snmp_notify_filter_subtree[SNMP_SIZE_OBJECTID];
    UINT32 subtree_len;
    UINT32 mask_len; /* no. of octets */
    INT32 snmp_notify_filter_type;
    UINT32 snmp_notify_filter_storage_type;
    UINT32 snmp_notify_filter_row_status;
    CHAR snmp_notify_filter_profile_name[MAX_FILTER_PROF_NAME_SIZE];
    UINT8 snmp_notify_filter_mask[MAX_FILTER_MASK_SIZE];
} SNMP_NOTIFY_FILTER_TABLE;
```

The members of the structure are defined in [Table 9-9](#). Some members of this structure have been omitted as they are used internally.

**Table 9-9. SNMP\_NOTIFY\_FILTER\_TABLE**

Member	Description
<code>snmp_notify_filter_subtree</code>	Sub-tree to which this filter is applied.
<code>subtree_len</code>	Length of the sub-tree.
<code>mask_len</code>	Number of octets in the mask.
<code>snmp_notify_filter_type</code>	Filter type. Valid values are: <ul style="list-style-type: none"><li>• INCLUDED – subtree family defined by this entry are accessible.</li><li>• EXCLUDED - subtree family defined by this entry are not accessible.</li></ul>



**Table 9-9. SNMP\_NOTIFY\_FILTER\_TABLE (cont.)**

Member	Description
snmp_notify_filter_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage.</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_notify_filter_row_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy.</li></ul> This is available only if Notification MIB is included in the build. The Notification MIB can be included in the build by setting the macro INCLUDE_MIB_NO to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
snmp_notify_filter_profile_name	Filter name with which this entry is associated.

**Table 9-9. SNMP\_NOTIFY\_FILTER\_TABLE (cont.)**

Member	Description
snmp_notify_filter_mask	<p>The bit mask which, in combination with the corresponding instance of snmpNotifyFilterSubtree, defines a family of subtrees which are included in or excluded from the Notify Filter Profile.</p> <p>Each bit of this bit mask corresponds to a sub-identifier of snmpNotifyFilterSubtree, with the most significant bit of the i-th octet of this octet string value corresponding to the (8i-7)-th sub-identifier. The least significant bit of the (8i)-th octet of this octet string corresponds to the (8i)-th sub-identifier. The range is 1 through 16.</p> <p>Each bit of this bit mask specifies whether or not the corresponding sub-identifiers must match when determining if an OBJECT IDENTIFIER matches this family of filter subtrees; a '1' indicates that an exact match must occur; a '0' indicates 'wild card', or any sub-identifier value is considered a match. The OBJECT IDENTIFIER X of an object instance is contained in a family of filter subtrees if, for each sub-identifier of the value of snmpNotifyFilterSubtree, either the i-th bit of snmpNotifyFilterMask is 0, or the i-th sub-identifier of X is equal to the i-th sub-identifier of the value of snmpNotifyFilterSubtree.</p> <p>If the value of this bit mask is M bits long and there are more than M sub-identifiers in the corresponding instance of snmpNotifyFilterSubtree, then the bit mask is extended with 1s to be the required length.</p>

## Related Topics

[SNMP Data Structures](#)

[SNMP\\_Add\\_To\\_Filter\\_Tbl](#)

[SNMP\\_Find\\_Filter\\_Entry](#)

[SNMP\\_Remove\\_From\\_Filter\\_Table](#)

## SNMP\_SS\_STRUCT

SNMP\_SS\_STRUCT is the typedef name for the snmp\_ss\_struct data structure. This data structure defines the Nucleus-supported [SNMP\\_Ss\\_Models](#) data object for users wishing to add a new Security Model. More detailed information on how to add a new Security Model is available in [SNMP Configurations](#).

```
typedef struct snmp_ss_struct
{
```

```
    UINT32      model;
    SNMP_INIT   snmp_init_cb;
    SNMP_CONFIG snmp_config_cb;
    SNMP_SECURE_OUTGOING_STRUCT process_outgoing_cb;
    SNMP_VERIFY_INCOMING_STRUCT process_incoming_cb;
} SNMP_SS_STRUCT;
```

The members of the structure are defined in [Table 9-10](#).

**Table 9-10. SNMP\_SS\_STRUCT**

Members	Description
model	Security model identifier.
snmp_init_cb	Callback function for initializing the model.
snmp_config_cb	Callback function for configuring the model.
process_outgoing_cb	Callback function for encrypting/signing the message before sending.
process_incoming_cb	Callback function for decrypting/authenticating incoming messages.

## Related Topics

[SNMP Data Structures](#)

[SNMP\\_Ss\\_Models](#)

[SNMP Configurations](#)

## SNMP\_TARGET\_ADDRESS\_TABLE

SNMP\_TARGET\_ADDRESS\_TABLE is the typedef name for the snmp\_target\_address\_table data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Target Address Table](#) data object.

```
typedef struct snmp_target_address_table
{
    UINT32 snmp_target_addr_tDomain;
    UINT32 snmp_target_addr_time_out;
    UINT32 tag_list_len;
    UINT32 params_len;
    INT32 snmp_target_addr_retry_count;
    UINT32 snmp_target_addr_mms;
    INT16 snmp_target_addr_tfamily;
    CHAR snmp_target_addr_name[MAX_TARG_NAME_SIZE];
    UINT8 snmp_target_addr_tAddress[SNMP_MAX_IP_ADDRS];
    UINT8 snmp_target_addr_tag_list[SNMP_SIZE_BUFCHR];
    UINT8 snmp_target_addr_params[MAX_TARG_NAME_SIZE];
    UINT8 snmp_target_addr_tmask[SNMP_MAX_IP_ADDRS];
    UINT8 snmp_ext_enabled;
    UINT8 snmp_target_addr_storage_type;
    UINT8 snmp_target_addr_row_status;
```

```
} SNMP_TARGET_ADDRESS_TABLE;
```

The members of the structure are defined in [Table 9-11](#). Some members of this structure have been omitted as they are used internally.

**Table 9-11. SNMP\_TARGET\_ADDRESS\_TABLE**

Member	Description
snmp_target_addr_tDomain	Target domain. Currently only UDP (1) is supported.
snmp_target_addr_timeout	Timeout in number of ticks. This value is used for timeouts when sending Notifications of type Inform. This value is not used currently.
tag_list_len	Length of tag list.
params_len	Params name length.
snmp_target_addr_retry_count	Number of retries when sending Notifications of type Inform. This value is not used currently.
snmp_target_addr_mms	Maximum message size value. (Part of the extended table). Value of 'zero(0)' indicates that maximum message size is unknown.
snmp_target_addr_tfamily	Target Address Family. Currently supported values are: <ul style="list-style-type: none"><li>• NU_FAMILY_IP</li><li>• NU_FAMILY_IP6.</li></ul>
snmp_target_addr_name	Target Address name. This uniquely identifies a Target Address entry.
snmp_target_addr_tAddress	Target IPv4 or IPv6 Address.
snmp_target_addr_tag_list	Tag list.
snmp_target_addr_params	Params name.
snmp_target_addr_tmaks	Target Address Mask. (Part of the extended table). This is used when snmp_ext_enabled has the value of NU_TRUE. Any SNMP Manager can use this entry that has IP Address masked to this value equal to the snmp_target_addr_tAddress masked to this value.
snmp_ext_enabled	NU_TRUE if the extended table is being used. NU_FALSE if the extended table is not being used. If its value is set to NU_TRUE then snmp_target_addr_tmask should be initialized.

Table 9-11. SNMP\_TARGET\_ADDRESS\_TABLE (cont.)

Member	Description
snmp_target_addr_storage_type	<p>Storage type. Valid values are:</p> <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage</li></ul> <p>This is available only if Target MIB is included in the build. The Target MIB can be included in the build by setting the macro INCLUDE_MIB_TARGET to NU_TRUE in <i>include/networking/snmp_cfg.h</i>. By default it is included in the build.</p>
snmp_target_addr_row_status	<p>Row status. Valid values are:</p> <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy</li></ul> <p>This is available only if Target MIB is included in the build. The Target MIB can be included in the build by setting the macro INCLUDE_MIB_TARGET to NU_TRUE in <i>include/networking/snmp_cfg.h</i>. By default it is included in the build.</p>

## Related Topics

[SNMP Data Structures](#)[SNMP\\_Add\\_Target](#)[SNMP\\_Find\\_Target](#)[SNMP\\_Remove\\_From\\_Tgr\\_Table](#)

## SNMP\_TARGET\_PARAMS\_TABLE

SNMP\_TARGET\_PARAMS\_TABLE is the typedef name for the snmp\_target\_params\_table data structure. This data structure defines the Nucleus-supported SNMPv3 [SNMP Target Params Table](#) data object.

```
typedef struct snmp_target_params_table
{
    UINT32 snmp_target_params_mp_model;
    UINT32 snmp_target_params_security_model;
    INT32  snmp_target_params_security_level;
    CHAR   snmp_target_params_security_name[SNMP_SIZE_BUFCHR];
    CHAR   snmp_target_params_name[MAX_TARG_NAME_SIZE];
    UINT8  snmp_target_params_storage_type;
    UINT8  snmp_target_params_row_status;
} SNMP_TARGET_PARAMS_TABLE;
```

The members of the structure are defined in [Table 9-12](#). Some members of this structure have been omitted as they are used internally.

**Table 9-12. SNMP\_TARGET\_PARAMS\_TABLE**

Member	Description
snmp_target_params_mp_model	Message Processing model identifier. It can have the following values: <ul style="list-style-type: none"><li>• SNMP_VERSION_V1 – SNMPv1</li><li>• SNMP_VERSION_V2 – SNMPv2c</li><li>• SNMP_VERSION_V3 – SNMPv3</li></ul>
snmp_target_params_security_model	Security Model. It can have the following values: <ul style="list-style-type: none"><li>• SNMP_CBSM_V1 – CBSMv1</li><li>• SNMP_CBSM_V2 – CBSMv2</li><li>• SNMP_USM – USM</li></ul>
snmp_target_params_security_level	Security level. Valid values are: <ul style="list-style-type: none"><li>• SNMP_SECURITY_NOAUTHNOPRIV</li><li>• SNMP_SECURITY_AUTHNOPRIV</li><li>• SNMP_SECURITY_AUTHPRIV</li></ul>
snmp_target_params_security_name	Security name. A Security to Group table entry should also exist that maps the security model and security name to a group; for that group there should be a VACM Access table entry through which access to MIB objects would be determined.
snmp_target_params_name	Params Name. This uniquely identifies an entry in the table.
snmp_target_params_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage</li></ul> This is available only if Target MIB is included in the build. The Target MIB can be included in the build by setting the macro INCLUDE_MIB_TARGET to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

**Table 9-12. SNMP\_TARGET\_PARAMS\_TABLE (cont.)**

Member	Description
snmp_target_params_row_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy.</li></ul> This is available only if Target MIB is included in the build. The Target MIB can be included in the build by setting the macro INCLUDE_MIB_TARGET to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

## Related Topics

[SNMP Data Structures](#)[SNMP\\_Add\\_Params](#)[SNMP\\_Find\\_Params](#)[SNMP\\_Remove\\_From\\_Params\\_Table](#)

## SNMP\_USM\_USER\_STRUCT

SNMP\_USM\_USER\_STRUCT is the typedef name for the snmp\_usm\_user\_struct data structure. This data structure defines the Nucleus-supported [Usm\\_User\\_Table](#) global array used to integrate USM users.

```
typedef struct snmp_usm_user_struct
{
    CHAR        usm_user_name[SNMP_SIZE_SMALLOBJECTID];
    UINT32      usm_auth_index;
    CHAR        usm_auth_password[USM_KEYCHANGE_MAX_SIZE];
    UINT32      usm_priv_index;
    CHAR        usm_priv_password[USM_KEYCHANGE_MAX_SIZE];
} SNMP_USM_USER_STRUCT;
```

The members of the structure are defined in [Table 9-13](#).

**Table 9-13. SNMP\_USM\_USER\_STRUCT**

Members	Description
usm_user_name	User Name. This value is used for the security name of this user. This value is also passed to VACM as a security name.

**Table 9-13. SNMP\_USM\_USER\_STRUCT (cont.)**

Members	Description
usm_auth_index	Authentication Protocol Index. This is the IANA assigned number for the Authentication Protocol. Nucleus SNMP supports HMAC-MD5 and HMAC-SHA Authentication Protocols. Macros defined for their authentication index are USM_MD5 and USM_SHA respectively. You may opt not to use Authentication Protocol by setting the value to USM_NOAUTH.
usm_auth_password	Authentication Password or Key depending upon the value of USM_PASSWORD_2_KEY. Refer to the <a href="#">Password to Key Utility</a> section for key generation.
usm_priv_index	Privacy Protocol Index. This is the IANA assigned number. USM only supports CBC-DES Privacy Protocol. The macro defining its value is USM_DES. You may opt not to use Privacy Protocol by setting its value to USM_NOPRIV.
usm_priv_password	Privacy Password or Key depending upon the value of USM_PASSWORD_2_KEY. Refer to the <a href="#">Password to Key Utility</a> section for key generation.

## Related Topics

[SNMP Data Structures](#)

[Usm\\_User\\_Table](#)

## USM\_USERS\_STRUCT

USM\_USERS\_STRUCT is the typedef name for the usm\_users\_struct data structure. This data structure defines the Nucleus-supported SNMPv3 [USM User Group](#) data object.

```
typedef struct usm_users_struct
{
    UINT32 usm_auth_index;
    UINT32 usm_priv_index;
    UINT8 usm_user_engine_id[SNMP_SIZE_SMALLOBJECTID];
    UINT8 usm_user_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 usm_security_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 usm_auth_key[USM_KEYCHANGE_MAX_SIZE];
    UINT8 usm_priv_key[USM_KEYCHANGE_MAX_SIZE];
    UINT8 usm_user_engine_id_len;
    UINT8 usm_storage_type;
    UINT8 usm_status;
} USM_USERS_STRUCT;
```



The members of the structure are defined in [Table 9-14](#). Some members of this structure have been omitted as they are used internally.

**Table 9-14. USM\_USERS\_STRUCT**

Member	Description
usm_auth_index	Authentication Protocol identifier. Nucleus SNMP supports HMAC-MD5 and HMAC-SHA Authentication Protocol. Authentication indexes are USM_MD5 for HMAC-MD5 and USM_SHA for HMAC-SHA. You can also use USM_NOAUTH for no Authentication Protocol.
usm_priv_index	Privacy Protocol identifier. Nucleus SNMP supports CBC-DES Privacy Protocol. The Privacy Protocol identifier for CBC-DES is USM_DES. You can also use USM_NOPRIV for no privacy service.
usm_user_engine_id	Engine ID.
usm_user_name	User name. It uniquely identifies an entry in the USM user table.
usm_security_name	Security name. This is the value passed by a SNMP Manager to the Nucleus SNMP Agent. It is convention in Nucleus SNMP to use the same value as usm_user_name. You can also use a different value. This value, along with security model (SNMP_USM), is used to select the Security for the Group VACM table.
usm_auth_key	Authentication key. This is the localized key. When a new password needs to be set, the localized key for that password can be created by invoking the callback function of the Authentication Protocol for a password to key conversion.
usm_priv_key	Privacy key. This is the localized key. When a new password needs to be set, the localized key for that password can be created by invoking the callback function of the Authentication Protocol for password to key conversion.
usm_user_engine_id_len	Engine ID length.

**Table 9-14. USM\_USERS\_STRUCT (cont.)**

Member	Description
usm_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage</li></ul> This is available only if USM MIB is included in the build. The USM MIB can be included in the build by setting the macro INCLUDE_MIB_USM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
usm_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy</li></ul> This is available only if USM MIB is included in the build. The USM MIB can be included in the build by setting the macro INCLUDE_MIB_USM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

## Related Topics

[SNMP Data Structures](#)

[USM\\_Add\\_User](#)

[USM\\_Lookup\\_Users](#)

[USM\\_Remove\\_User](#)

[USM\\_Save\\_User](#)

## USM\_PRIV\_PROT\_STRUCT

USM\_PRIV\_PROT\_STRUCT is the typedef name for the usm\_priv\_prot\_struct data structure. This data structure defines the Nucleus-supported [Usm\\_Priv\\_Prot\\_Table](#) global array that integrates Privacy Protocols into the USM.

```
typedef struct usm_priv_prot_struct
{
    UINT32          usm_index;
    USM_ENCRYPT_STRUCT usm_encrypt_cb;
    USM_DECRYPT_STRUCT usm_decrypt_cb;
    UINT8          usm_param_length;
} USM_PRIV_PROT_STRUCT;
```

The members of the structure are defined in [Table 9-15](#). Some members of this structure have been omitted as they are used internally.

**Table 9-15. USM\_PRIV\_PROT\_STRUCT**

Member	Description
usm_index	Privacy Protocol identifier. Nucleus SNMP supports CBC-DES Privacy Protocol and noPriv protocol. The Privacy Protocol identifier for CBC-DES is USM_DES and for noPriv Protocol USM_NOPRIV.
usm_encrypt_cb	Callback function for encrypting an outgoing message.
usm_decrypt_cb	Callback function for decrypting an incoming message.
usm_param_length	Privacy parameter's length.

## Related Topics

[SNMP Data Structures](#)

[USM\\_Lookup\\_Priv\\_Prot](#)

[Usm\\_Priv\\_Prot\\_Table](#)

## USM\_AUTH\_PROT\_STRUCT

USM\_AUTH\_PROT\_STRUCT is the typedef name for the `usm_auth_prot_struct` data structure. This data structure defines the Nucleus-supported [Usm\\_Auth\\_Prot\\_Table](#) global array that integrates the Authentication Protocols into the USM.

```
typedef struct usm_auth_prot_struct
{
    UINT32                usm_index;
    USM_AUTH_OUTGOING_STRUCT usm_secure_cb;
    USM_AUTH_INCOMING_STRUCT usm_verify_cb;
    USM_HASH_STRUCT        usm_password_cb;
    USM_KEY_CHANGE          usm_key_change_cb;
    UINT32                  usm_param_length;
    UINT8                   key_length;
} USM_AUTH_PROT_STRUCT;
```

The members of the structure are defined in [Table 9-16](#). Some members of this structure have been omitted as they are used internally.

**Table 9-16. USM\_AUTH\_PROT\_STRUCT**

Member	Description
usm_index	Authentication Protocol identifier.
usm_secure_cb	Callback function for signing an outgoing message.
usm_verify_cb	Callback function for authenticating an incoming message.

**Table 9-16. USM\_AUTH\_PROT\_STRUCT (cont.)**

Member	Description
usm_password_cb	Callback function for calculating the hash of a new password. This will compute the localized key.
usm_key_change_cb	Callback function for changing a key as defined in USM MIB.
usm_param_length	Authentication parameter's length.
key_length	Length of the key.

## Related Topics

[SNMP Data Structures](#)

[USM\\_Lookup\\_Auth\\_Prot](#)

[Usm\\_Auth\\_Prot\\_Table](#)

## VACM\_CONTEXT\_STRUCT

VACM\_CONTEXT\_STRUCT is the typedef name for the vacm\_context\_struct data structure.

```
typedef struct vacm_context_struct
{
    CHAR context_name[SNMP_SIZE_SSMALLOBJECTID];
}VACM_CONTEXT_STRUCT;
```

The members of the structure are defined in [Table 9-17](#). Some members of this structure have been omitted as they are used internally.

**Table 9-17. VACM\_CONTEXT\_STRUCT**

Member	Description
context_name	A human readable name identifying a particular context at a particular SNMP entity. The empty contextName (zero length) represents the default context.

## Related Topics

[SNMP Data Structures](#)

[VACM\\_Search\\_Context](#)

## VACM\_SEC2GROUP

VACM\_SEC2GROUP is the typedef name for the vacm\_sec2group data structure. This data structure defines the Nucleus-supported [Snmp\\_Cfg\\_Sec2Groups](#) global array that holds the Security to Group entries to be configured during initialization.

```
typedef struct vacm_sec2group
{
    UINT32      vacm_security_model;
    UINT8       vacm_security_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8       vacm_group_name[SNMP_SIZE_SMALLOBJECTID];

}VACM_SEC2GROUP;
```

The members of the structure are defined in [Table 9-18](#).

**Table 9-18. VACM\_SEC2GROUP**

Members	Description
vacm_security_model	The Security Model, this may not take the value of ‘any’ (0).
vacm_security_name	The Security Name for the principal, represented in Security Model independent format, which is mapped by this entry to a group name.
vacm_group_name	The name of the group to which this entry belongs. That is the name of the group to which the combination of security model and security name is mapped. This group name is used as index into the VACM Access table to select an access policy.

## Related Topics

[SNMP Data Structures](#)

[Snmplib\\_Sec2Groups](#)

## VACM\_SEC2GROUP\_STRUCT

VACM\_SEC2GROUP\_STRUCT is the typedef name for the vacm\_sec2group\_struct data structure. This data structure defines the Nucleus-supported SNMPv3 [VACM Security To Group](#) data object.

```
typedef struct vacm_sec2group_struct
{
    UINT32 vacm_security_model;
    UINT8  vacm_security_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8  vacm_group_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8  vacm_storage_type;
    UINT8  vacm_status;
}VACM_SEC2GROUP_STRUCT;
```

The members of the structure are defined in [Table 9-19](#). Some members of this structure have been omitted as they are used internally.

**Table 9-19. VACM\_SEC2GROUP\_STRUCT**

Member	Description
vacm_security_model	Security model identifier. It can have the following values: <ul style="list-style-type: none"> <li>• SNMP_CBSM_V1 – CBSMv1</li> <li>• SNMP_CBSM_V2 – CBSMv2</li> <li>• SNMP_USM – USM</li> </ul>
vacm_security_name	Security name.
vacm_group_name	Group name.
vacm_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"> <li>• SNMP_STORAGE_OTHER – other storage</li> <li>• SNMP_STORAGE_VOLATILE – volatile storage</li> <li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li> <li>• SNMP_STORAGE_PERMANENT – permanent storage</li> <li>• SNMP_STORAGE_READONLY – read-only storage.</li> </ul> This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro INCLUDE_MIB_VACM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
vacm_status	Row status. Valid values are: <ul style="list-style-type: none"> <li>• SNMP_ROW_ACTIVE - active</li> <li>• SNMP_ROW_NOTINSERVICE – not in service</li> <li>• SNMP_ROW_NOTREADY – not ready to use</li> <li>• SNMP_ROW_DESTROY - destroy</li> </ul> This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro INCLUDE_MIB_VACM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

## Related Topics

[SNMP Data Structures](#)[VACM\\_InsertGroup](#)[VACM\\_Search\\_Group](#)[VACM\\_Remove\\_Group](#)[VACM\\_Save\\_Group](#)

## VACM\_ACCESS

VACM\_ACCESS is the typedef name for the vacm\_access data structure. This data structure defines the Nucleus-supported [Snmpp\\_Cfg\\_Access](#) global array that holds entries that define the read, write and notify access based upon security model, group name, context prefix, and security level.

```
typedef struct vacm_access
{
    UINT32      vacm_security_model;
    UINT8       vacm_group_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8       vacm_context_prefix[SNMP_SIZE_SMALLOBJECTID];
    UINT8       vacm_context_match;
    UINT8       vacm_security_level;
    UINT8       vacm_read_view[SNMP_SIZE_SMALLOBJECTID];
    UINT8       vacm_write_view[SNMP_SIZE_SMALLOBJECTID];
    UINT8       vacm_notify_view[SNMP_SIZE_SMALLOBJECTID];
} VACM_ACCESS;
```

The members of the structure are defined in [Table 9-20](#). Some members of this structure have been omitted as they are used internally.

**Table 9-20. VACM\_ACCESS**

Members	Description
vacm_security_model	The Security Model. This may take the value of ‘any’ (SNMP_ANY). Nucleus SNMP supports three security models CBSMv1 (SNMP_CBSM_V1), CBSMv2 (SNMP_CBSM_V2) and USM (SNMP_USM).
vacm_group_name	Group name for this entry.
vacm_context_prefix	To gain access rights allowed by this entry: <ul style="list-style-type: none"><li>• The context name must match exactly with the value of this variable if the value of vacm_context_match is ‘exact’ (1 or VACM_CTXMATCH_EXACT).</li><li>• The context name must partially match the value of this variable if the value of vacm_context_match is ‘prefix’ (2 or VACM_CTXMATCH_PREFIX).</li></ul>

**Table 9-20. VACM\_ACCESS (cont.)**

Members	Description
vacm_context_match	This defines how the value of the context should match, whether the value should match exact or is a prefix (should match partially). Valid values are: <ul style="list-style-type: none"><li>• VACM_CTXMATCH_EXACT</li><li>• VACM_CTXMATCH_PREFIX</li></ul>
vacm_security_level	This member defines the minimum security level required by this access entry. A security level of noAuthNoPriv is less than authNoPriv which in turn is less than authPriv.
vacm_read_view	Read MIB view.
vacm_write_view	Write MIB view.
vacm_notify_view	Notify MIB view.

## Related Topics

[SNMP Data Structures](#)

[Snmplib\\_Cfg\\_Access](#)

## VACM\_ACCESS\_STRUCT

VACM\_ACCESS\_STRUCT is the typedef name for the vacm\_access\_struct data structure. This data structure defines the Nucleus-supported SNMPv3 [VACM Access Group](#) data object.

```
typedef struct vacm_access_struct
{
    UINT32 vacm_security_model;
    UINT8 vacm_group_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_context_prefix[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_security_level;
    UINT8 vacm_context_match;
    UINT8 vacm_read_view[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_write_view[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_notify_view[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_storage_type;
    UINT8 vacm_status;
} VACM_ACCESS_STRUCT;
```



The members of the structure are defined in [Table 9-21](#). Some members of this structure have been omitted as they are used internally.

**Table 9-21. VACM\_ACCESS\_STRUCT**

Member	Description
vacm_security_model	Security model identifier. It can have the following values: <ul style="list-style-type: none"><li>• SNMP_CBSM_V1 – CBSMv1</li><li>• SNMP_CBSM_V2 – CBSMv2</li><li>• SNMP_USM – USM</li></ul>
vacm_group_name	Group name. A Group is a set of zero or more <security model, security name> tuples on whose behalf SNMP management objects can be accessed. A group defines the access rights afforded to all security names which belong to that group. The combination of security model and security name maps to at most one group.
vacm_context_prefix	Context prefix. In order to gain access rights allowed by this entry, the context name: <ul style="list-style-type: none"><li>• must match exactly if the value of vacm_context_match is 'exact' (VACM_CTXMATCH_EXACT)</li><li>• must patch partially if the value of vacm_context_match is 'prefix' (VACM_CTXTMATCH_PREFIX)</li></ul>
vacm_security_level	Security level. Valid values are: <ul style="list-style-type: none"><li>• SNMP_SECURITY_NOAUTHNOPRIV – no authentication and no privacy.</li><li>• SNMP_SECURITY_AUTHNOPRIV – authentication and no privacy.</li><li>• SNMP_SECURITY_AUTHPRIV – authentication with privacy.</li></ul>
vacm_context_match	Context match. Valid values are: <ul style="list-style-type: none"><li>• VACM_CTXTMATCH_EXACT</li><li>• VACM_CTXTMATCH_PREFIX</li></ul>
vacm_read_view	Read view. On validating read access, the Nucleus SNMP Agent searches this view name in the VACM MIB View table and validates whether a particular managed object is accessible by this view.
vacm_write_view	Write view. On validating write access, the Nucleus SNMP Agent searches this view name in the VACM MIB View table and validates whether a particular managed object is accessible by this view.
vacm_notify_view	Notify view. On validating notify access, the Nucleus SNMP Agent searches this view name in the VACM MIB View table and validates whether a particular managed object is accessible by this view.

**Table 9-21. VACM\_ACCESS\_STRUCT (cont.)**

Member	Description
vacm_storage_type	Storage type. Valid values are: <ul style="list-style-type: none"><li>• SNMP_STORAGE_OTHER – other storage</li><li>• SNMP_STORAGE_VOLATILE – volatile storage</li><li>• SNMP_STORAGE_NONVOLATILE – non-volatile storage</li><li>• SNMP_STORAGE_PERMANENT – permanent storage</li><li>• SNMP_STORAGE_READONLY – read-only storage</li></ul> This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro INCLUDE_MIB_VACM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.
vacm_status	Row status. Valid values are: <ul style="list-style-type: none"><li>• SNMP_ROW_ACTIVE - active</li><li>• SNMP_ROW_NOTINSERVICE – not in service</li><li>• SNMP_ROW_NOTREADY – not ready to use</li><li>• SNMP_ROW_DESTROY - destroy</li></ul> This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro INCLUDE_MIB_VACM to NU_TRUE in <i>include/networking/snmp_cfg.h</i> . By default it is included in the build.

## Related Topics

[SNMP Data Structures](#)

[VACM\\_InsertAccessEntry](#)

[VACM\\_Search\\_AccessEntry](#)

[VACM\\_Remove\\_AccessEntry](#)

[VACM\\_Save\\_Access](#)

## VACM\_VIEWTREE

VACM\_VIEWTREE is the typedef name for the vacm\_viewtree data structure. This data structure defines the Nucleus-supported [Snmplib\\_Mib\\_View](#) global array that holds entries that define which MIB objects are included in and are excluded from a MIB VIEW.

```
typedef struct vacm_viewtree
{
    UINT8      vacm_view_name[SNMP_SIZE_SMALLOBJECTID];
    UINT32     vacm_subtree[VACM_SUBTREE_LEN];
    UINT32     vacm_subtree_len;
    UINT8      vacm_family_mask[VACM_MASK_SIZE];
    UINT32     vacm_mask_length;
```

```
    UINT8      vacm_family_type;  
}VACM_VIEWTREE;
```

The members of the structure are defined in [Table 9-22](#).

**Table 9-22. VACM\_VIEWTREE**

Members	Description
vacm_view_name	Human readable name for a family of view subtrees.
vacm_subtree	Object Identifier of subtree.
vacm_subtree_len	Length of object identifier of subtree.
vacm_family_mask	<p>The bit mask which, in combination with the corresponding value of vacm_subtree, defines a family of view subtrees.</p> <p>Each bit of this bit mask corresponds to a sub-identifier of vacmViewTreeFamilySubtree, with the most significant bit of the i-th octet of this octet string value corresponding to the (8i-7)-th sub-identifier, and the least significant bit of the i-th octet of this octet string corresponding to the (8i)-th sub-identifier. The range is 1 through 16.</p> <p>Each bit of this bit mask specifies whether or not the corresponding sub-identifiers must match when determining if an OBJECT IDENTIFIER is in this family of view subtrees. A value of '1' indicates that an exact match must occur; a '0' indicates 'wild card' (any sub-identifier value will match).</p> <p>The OBJECT IDENTIFIER X of an object instance is contained in a family of view subtrees if, for each sub-identifier of the value of vacm_subtree, either: the -th bit of vacm_family_mask is 0, or the -th sub-identifier of X is equal to the -th sub-identifier of the value of vacm_subtree.</p> <p>If the value of this bit mask is M bits long and there are more than M sub-identifiers in the corresponding instance of vacmViewTreeFamilySubtree, then the bit mask is extended with ones to be the required length.</p>
vacm_mask_length	Length of mask in bytes.
vacm_family_type	Family Type. The valid values are included (1) and excluded (2).

## Related Topics

[SNMP Data Structures](#)

[Snm\\_Cfg\\_Mib\\_View](#)

## VACM\_VIEWTREE\_STRUCT

VACM\_VIEWTREE\_STRUCT is the typedef name for the vacm\_viewtree\_struct data structure. This data structure defines the Nucleus-supported SNMPv3 [VACM View Tree Family Group](#) data object.

```
typedef struct vacm_viewtree_struct
{
    UINT32 vacm_subtree[VACM_SUBTREE_LEN];
    UINT32 vacm_subtree_len;
    UINT32 vacm_mask_length;
    UINT8 vacm_view_name[SNMP_SIZE_SMALLOBJECTID];
    UINT8 vacm_family_mask[VACM_MASK_SIZE];
    UINT8 vacm_family_type;
    UINT8 vacm_storage_type;
    UINT8 vacm_status;
} VACM_VIEWTREE_STRUCT;
```

The members of the structure are defined in [Table 9-23](#). Some members of this structure have been omitted as they are used internally.

**Table 9-23. VACM\_VIEWTREE\_STRUCT**

Member	Description
vacm_subtree	Sub-tree for the view.
vacm_subtree_len	Sub-tree length.
vacm_mask_length	Mask length in octets.
vacm_view_name	View name.

**Table 9-23. VACM\_VIEWTREE\_STRUCT (cont.)**

Member	Description
vacm_family_mask	<p>Family mask. The bit mask which, in combination with the corresponding instance of vacmView TreeFamilySubtree, defines a family of view subtrees.</p> <p>Each bit of this bit mask corresponds to a sub-identifier of vacmView TreeFamilySubtree, with the most significant bit of the i-th octet of this octet string value corresponding to the (8i-7)-th sub-identifier, and the least significant bit of the i-th octet of this octet string corresponding to the (8i)-th sub-identifier. The range is 1 through 16.</p> <p>Each bit of this bit mask specifies whether or not the corresponding sub-identifiers must match when determining if an OBJECT IDENTIFIER is in this family of view subtrees; a '1' indicates that an exact match must occur; a '0' indicates 'wild card', for example, any sub-identifier value matches.</p> <p>Thus, the OBJECT IDENTIFIER X of an object instance is contained in a family of view subtrees if, for each sub-identifier of the value of vacm ViewTreeFamily Subtree, either the -th bit of vacmView TreeFamily Mask is 0, or the -th sub-identifier of X is equal to the i-th sub-identifier of the value of vacmViewTree FamilySub tree.</p> <p>If the value of this bit mask is M bits long and there are more than M sub-identifiers in the corresponding instance of vacm ViewTreeFamily Subtree, then the bit mask is extended with "1" values to be the required length.</p>
vacm_family_type	<p>Family type indicates whether the corresponding family of subtrees defined by vacm_subtree and vacm_family_mask are accessible or not through the view named vacm_view_name. Valid values are:</p> <ul style="list-style-type: none"><li>• VACM_FAMILY_INCLUDED – The family of subtrees is accessible through the view.</li><li>• VACM_FAMILY_EXCLUDED – The family of subtrees is not accessible through the view.</li></ul>

**Table 9-23. VACM\_VIEWTREE\_STRUCT (cont.)**

Member	Description
<code>vacm_storage_type</code>	<p>Storage type. Valid values are:</p> <ul style="list-style-type: none"><li>• <code>SNMP_STORAGE_OTHER</code> – other storage</li><li>• <code>SNMP_STORAGE_VOLATILE</code> – volatile storage</li><li>• <code>SNMP_STORAGE_NONVOLATILE</code> – non-volatile storage</li><li>• <code>SNMP_STORAGE_PERMANENT</code> – permanent storage</li><li>• <code>SNMP_STORAGE_READONLY</code> – read-only storage</li></ul> <p>This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro <code>INCLUDE_MIB_VACM</code> to <code>NU_TRUE</code> in <i>include/networking/snmp_cfg.h</i>. By default it is included in the build.</p>
<code>vacm_status</code>	<p>Row status. Valid values are:</p> <ul style="list-style-type: none"><li>• <code>SNMP_ROW_ACTIVE</code> - active</li><li>• <code>SNMP_ROW_NOTINSERVICE</code> – not in service</li><li>• <code>SNMP_ROW_NOTREADY</code> – not ready to use</li><li>• <code>SNMP_ROW_DESTROY</code> - destroy</li></ul> <p>This is available only if VACM MIB is included in the build. The VACM MIB can be included in the build by setting the macro <code>INCLUDE_MIB_VACM</code> to <code>NU_TRUE</code> in <i>include/networking/snmp_cfg.h</i>. By default it is included in the build.</p>

## Related Topics

[SNMP Data Structures](#)

[VACM\\_InsertMibView](#)

[VACM\\_Search\\_MibView](#)

[VACM\\_Remove\\_MibView](#)

[VACM\\_Save\\_View](#)

## SNMP API Functions

This section describes the services that are provided by Nucleus SNMP to applications running on the same target. To invoke any of the API functions, *snmp\_api.h* must be included. Note that all API functions (that add a node to a list) allocate memory for the node within the function. Therefore, the pointers passed to the API functions can be pointing to objects with local scope.

- [NU\\_SNMP\\_Initialize](#)
- [SNMP\\_Configuration](#)
- [NU\\_SNMP\\_Set\\_Host\\_Id](#)

- `NU_SNMP_Set_Host_Id6`
- `SNMP_Mib_Register`
- `SNMP_Mib_Unregister`
- `SNMP_Get_Notification_Ptr`
- `SNMP_Notification_Ready`
- `AgentGetAuthenTraps`
- `AgentSetAuthenTraps`
- `AgentSetColdTraps`
- `SNMP_Get_Engine_ID`
- `CBSM_Add_Community`
- `CBSM_Find_Community_Index`
- `CBSM_Remove_Community`
- `CBSM_Save_Community`
- `SNMP_Add_To_Notify_Tbl`
- `SNMP_Find_Notify_Entry`
- `SNMP_Remove_From_Notify_Table`
- `SNMP_Add_To_Profile_Tbl`
- `SNMP_Find_Profile_Entry`
- `SNMP_Remove_From_Profile_Table`
- `SNMP_Add_To_Filter_Tbl`
- `SNMP_Find_Filter_Entry`
- `SNMP_Remove_From_Filter_Table`
- `SNMP_Add_Target`
- `SNMP_Find_Target`
- `SNMP_Remove_From_Tgr_Table`
- `SNMP_Add_Params`
- `SNMP_Find_Params`
- `SNMP_Remove_From_Params_Table`
- `USM_Add_User`

- [USM\\_Lookup\\_Users](#)
- [USM\\_Remove\\_User](#)
- [USM\\_Save\\_User](#)
- [USM\\_Lookup\\_Priv\\_Prot](#)
- [USM\\_Lookup\\_Auth\\_Prot](#)
- [VACM\\_Add\\_Context](#)
- [VACM\\_Search\\_Context](#)
- [VACM\\_Remove\\_Context](#)
- [VACM\\_InsertGroup](#)
- [VACM\\_Search\\_Group](#)
- [VACM\\_Remove\\_Group](#)
- [VACM\\_Save\\_Group](#)
- [VACM\\_InsertAccessEntry](#)
- [VACM\\_Search\\_AccessEntry](#)
- [VACM\\_Remove\\_AccessEntry](#)
- [VACM\\_Save\\_Access](#)
- [VACM\\_InsertMibView](#)
- [VACM\\_Search\\_MibView](#)
- [VACM\\_Remove\\_MibView](#)
- [VACM\\_Save\\_View](#)



## NU\_SNMP\_Initialize

This function initializes Nucleus SNMP. All subsystems and models as well as the MIB are initialized. All tasks required by Nucleus SNMP are created and started here. Nucleus NET invokes this function.

### Usage

```
STATUS NU_SNMP_Initialize (VOID);
```

### Return Values

- **NU\_SUCCESS**  
Nucleus SNMP initialized successfully.
- **NU\_NO\_MEMORY**  
Memory Allocation failed.

### Related Topics

[SNMP API Functions](#)

[SNMP\\_Configuration](#)

## SNMP\_Configuration

If initialization fails, Nucleus SNMP enters configuration mode. All subsystems and models which failed to initialize are configured in this function. This routine must be called after the Networking support (networking stack and networking interfaces) are available.

Initialization fails, when Nucleus SNMP is unable to determine its previous state. A real file system is required for Nucleus SNMP to remember its previous state. Therefore, if a real file system is not used, initialization will always fail and configuration is required.

### Usage

```
VOID SNMP_Configuration (VOID);
```

### Example

```
NU_IOCTL_OPTION      device_ip;

/* Initialize file system here if it is there. */
.
.
.

/* Wait until the Networking support is available. */
NETBOOT_Wait_For_Network_Up(NU_SUSPEND);

/* Point to the name of the device of which you wish to know the IP
 * Address. This can be obtained from current.imageconfig.html
 * located under: /output/toolchain/<platform>/debug directory
 * of your system project.
 */
device_ip.s_optval = (UINT8*) DEMOI_Device_Name;

/* Call NU_Ioctl to get the IP address. */
NU_Ioctl (IOCTL_GETIFADDR, &device_ip, sizeof(NU_IOCTL_OPTION));

/* Copy the retrieved IP address. */
memcpy (DEMOI_Local_Ip, device_ip.s_ret.s_ipaddr, 4);

/* Set the local IP for the SNMP Engine ID. */
NU_SNMP_Set_Host_Id (DEMOI_Local_Ip);

/* SNMP modules may need to be configured, if initialization was
 * not successful (possibly because SNMP is being run for the
 * first time).
 */
SNMP_Configuration ();
```

### Related Topics

[SNMP API Functions](#)

[NU\\_SNMP\\_Initialize](#)

## NU\_SNMP\_Set\_Host\_Id

This function sets the internal variable, `cfg_hostid`, with the local IPv4 address located at `ipaddr`. During initialization the SNMP Engine ID is initialized to a value that contains the IP address in `cfg_hostid`.

### Usage

```
STATUS NU_SNMP_Set_Host_Id (UINT8 *ipaddr);
```

### Arguments

- `ipaddr`  
A pointer to the local IPv4 address of the interface.

### Return Values

- `NU_SUCCESS`  
Host ID set successfully.

### Example

```
NU_IOCTL_OPTION    device_ip;

/* Initialize file system here if it is there. */
.
.
.

/* Wait until the Networking support is available. */
NETBOOT_Wait_For_Network_Up(NU_SUSPEND);

/* Point to the name of the device of which you wish to know the IP
 * Address. This can be obtained from current.imageconfig.html
 * located under: /output/toolchain/<platform>/debug directory
 * of your system project.
 */
device_ip.s_optval = (UINT8*) DEMOI_Device_Name;

/* Call NU_Ioctl to get the IP address. */
NU_Ioctl (IOCTL_GETIFADDR, &device_ip, sizeof(NU_IOCTL_OPTION));

/* Copy the retrieved IP address. */
memcpy (DEMOI_Local_Ip, device_ip.s_ret.s_ipaddr, 4);

/* Set the local IP for the SNMP Engine ID. */
NU_SNMP_Set_Host_Id (DEMOI_Local_Ip);

/* SNMP modules may need to be configured, if initialization was
 * not successful (possibly because SNMP is being run for the
 * first time).
 */
SNMP_Configuration ();
```

## Related Topics

[SNMP API Functions](#)

[NU\\_SNMP\\_Set\\_Host\\_Id6](#)

## NU\_SNMP\_Set\_Host\_Id6

This function sets the internal variable, `cfig_hostid`, with the local IPv6 address located at `ipaddr`. During initialization the SNMP Engine ID is initialized to a value that contains the IP address in `cfig_hostid`.

### Usage

```
STATUS SNMP_Set_Host_Id6 (UINT8 *ipaddr);
```

### Arguments

- `ipaddr`  
A pointer to the local IPv6 address of the interface.

### Return Values

- `NU_SUCCESS`  
Host ID set successfully.

### Example

```
NU_IOCTL_OPTION    device_ip;

/* Initialize file system here if it is there. */
.
.
.
/* Wait until the Networking support is available. */
NETBOOT_Wait_For_Network_Up(NU_SUSPEND);

/* Point to the name of the device of which you wish to know the IP
 * Address. This can be obtained from current.imageconfig.html
 * located under: /output/toolchain/<platform>/debug directory
 * of your system project.
 */
device_ip.s_optval = (UINT8*) DEMOI_Device_Name;

/* Call NU_Ioctl to get the IP address. */
NU_Ioctl (SIOCGIFADDR_IN6, &device_ip, sizeof(NU_IOCTL_OPTION));

/* Set the local IP for the SNMP Engine ID. */
NU_SNMP_Set_Host_Id6 (device_ip.s_ret.s_ipaddr);

/* SNMP modules may need to be configured, if initialization was
 * not successful (possible because SNMP is being run for the
 * first time).
 */
SNMP_Configuration ();
```

### Related Topics

[SNMP API Functions](#)

[NU\\_SNMP\\_Set\\_Host\\_Id](#)

## SNMP\_Mib\_Register

This function registers MIB with Nucleus SNMP.

### Usage

```
STATUS SNMP_Mib_Register (mib_element_t *mib,  
                          UINT16      mibsize);
```

### Arguments

- **mib**  
Pointer to the array of MIB elements. This array should be globally or dynamically allocated. Nucleus SNMP holds a pointer to the array. Nucleus SNMP uses the same memory instead of allocating memory for a copy. If memory pointed by this pointer gets deallocated, it will result in unpredictable behavior of SNMP. It may lead the system to crash.  
  
Data structure [mib\\_element\\_t](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- **mibsize**  
Size of array pointer by mib or the number of MIB elements pointed to by the mib.

### Return Values

- **NU\_SUCCESS**  
MIBs are registered successfully.
- **NU\_NO\_MEMORY**  
Memory allocation failed.
- **NU\_INVALID**  
At least one of the MIB objects is already registered.

### Example

The following example demonstrates the registration of IP Tunnel MIB with SNMP.

```
/* Global Variable */  
mib_element_t IP_Tun_Mib [] =  
{  
    #include "networking/ip_tun_oid.h"  
};  
  
.  
.  
.  
  
/* Register IP tunnel MIB with SNMP. */  
status = SNMP_Mib_Register (IP_Tun_Mib,  
                           sizeof(IP_Tun_Mib) / sizeof(mib_element_t));
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Mib\\_Unregister](#)

[mib\\_element\\_t](#)

## SNMP\_Mib\_Unregister

This function unregisters MIBs from Nucleus SNMP.

### Usage

```
STATUS SNMP_Mib_Register (mib_element_t *mib,  
                          UINT16      mibsize);
```

### Arguments

- **mib**  
Pointer to the array of MIB elements. This array should be globally or dynamically allocated. Nucleus SNMP holds pointer to it. Nucleus SNMP uses the same memory instead of allocating memory for a copy. If memory pointed to by this pointer gets deallocated, it will result in unpredictable behavior of SNMP. It may lead the system to crash.  
  
Data structure [mib\\_element\\_t](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- **mibsize**  
Size of array pointer by mib or the number of MIB elements pointed by the mib.

### Return Values

- **NU\_SUCCESS**  
MIBs are successfully unregistered.

### Example

The following example demonstrates the registration of an IP Tunnel MIB with SNMP.

```
/* Global Variable */  
mib_element_t IP_Tun_Mib [] =  
{  
    #include "networking/ip_tun_oid.h"  
};  
.  
.  
.  
/* Un-register IP tunnel MIB with SNMP. */  
status = SNMP_Mib_Unregister (IP_Tun_Mib,  
                             sizeof(IP_Tun_Mib) / sizeof(mib_element_t));
```

### Related Topics

[SNMP API Functions](#)

[SNMP\\_Mib\\_Register](#)

[SNMP Data Structures](#)

[mib\\_element\\_t](#)



## SNMP\_Get\_Notification\_Ptr

This function gets a notification request structure pointer from a pool of pointers. This structure is filled with the information of the notification to be sent.

### Usage

```
STATUS SNMP_Get_Notification_Ptr (SNMP_NOTIFY_REQ_STRUCT **notification);
```

### Arguments

- notification

Double pointer to the notification to be sent.

Data structure [CBSM\\_COMMUNITY\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Notification pointer obtained successfully.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Example

The following is an example for generating notifications. Note that *snmp\_api.h* is mandatory.

```
/* Handle to the notification request structure */
SNMP_NOTIFY_REQ_STRUCT *snmp_notification;

/* OID of 'ipv6IfStateChange' */
UINT32                ipv6_if_state_change_oid[] =
                        IPV6_IF_STATE_CHANGE_OID;

/* OID of 'ipv6IfDescr' */
UINT32                ipv6_if_descr_oid[] =
                        IPV6_IF_DESCR_OID;

/* OID of 'ipv6IfOperStatus' */
UINT32                ipv6_if_oper_status_oid[] =
                        IPV6_IF_OPER_STATUS_OID;

/* If you successfully got the handle to the notification */
if (SNMP_Get_Notification_Ptr (&snmp_notification) ==
    NU_SUCCESS)
{
    /* Clear out the notification structure. */
    memset (snmp_notification, 0,
            sizeof(SNMP_NOTIFY_REQ_STRUCT));

    /* Update the notification OID. */
    memcpy (snmp_notification->OID.notification_oid,
            ipv6_if_state_change_oid,
```

```
        sizeof(ipv6_if_state_change_oid));

/* Update the notification OID length. */
snmp_notification->OID.oid_len =
    IPV6_IF_STATE_CHANGE_OID_LEN;

/* Set first bound object as 'ipv6IfDescr'. */
memcpy (snmp_notification->snmp_object_list[0].Id,
        ipv6_if_descr_oid,
        sizeof(ipv6_if_descr_oid));

/* Set interface index in OID. */
snmp_notification->snmp_object_list[0].
    Id[(IPV6_IF_DESCR_OID_LEN - 1)] = 1;

/* Set OID length. */
snmp_notification->snmp_object_list[0].IdLen =
    IPV6_IF_DESCR_OID_LEN;

/* Set second bound object as 'ipv6IfOperStatus'. */
memcpy (snmp_notification->snmp_object_list[1].Id,
        ipv6_if_oper_status_oid,
        sizeof(ipv6_if_oper_status_oid));

/* Update the interface index in the OID. */
snmp_notification->snmp_object_list[1].
    Id[(IPV6_IF_OPER_STATUS_OID_LEN - 1)] = 1;

/* Set OID length. */
snmp_notification->snmp_object_list[1].IdLen =
    IPV6_IF_OPER_STATUS_OID_LEN;

/* Update the number of bound objects. */
snmp_notification->snmp_object_list_len = 2;

/* Send the notification. */
SNMP_Notification_Ready (snmp_notification);
}
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Notification\\_Ready](#)

[CBSM\\_COMMUNITY\\_STRUCT](#)

## SNMP\_Notification\_Ready

This function indicates to the Notification Originator Application that the notification request structure has been filled and is ready to be sent.

### Usage

```
STATUS SNMP_Notification_Ready (SNMP_NOTIFY_REQ_STRUCT *notification);
```

### Arguments

- notification

Pointer to the notification to be sent.

Data structure [CBSM\\_COMMUNITY\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

Notification or trap is registered to be sent by Notification Originator Task.

- SNMP\_ERROR

Null pointer passed in.

### Example

The following is an example for generating notifications. Note that *snmp\_api.h* is mandatory.

```
/* Handle to the notification request structure. */
SNMP_NOTIFY_REQ_STRUCT *snmp_notification;

/* OID of 'ipv6IfStateChange'. */
UINT32          ipv6_if_state_change_oid[] =
                IPV6_IF_STATE_CHANGE_OID;

/* OID of 'ipv6IfDescr'. */
UINT32          ipv6_if_descr_oid[] =
                IPV6_IF_DESCR_OID;

/* OID of 'ipv6IfOperStatus'. */
UINT32          ipv6_if_oper_status_oid[] =
                IPV6_IF_OPER_STATUS_OID;

/* If you successfully got the handle to the notification. */
if (SNMP_Get_Notification_Ptr (&snmp_notification) ==
                                NU_SUCCESS)
{
    /* Clear out the notification structure. */
    memset (snmp_notification, 0,
            sizeof(SNMP_NOTIFY_REQ_STRUCT));

    /* Update the notification OID. */
    memcpy (snmp_notification->OID.notification_oid,
            ipv6_if_state_change_oid,
```

```
        sizeof(ipv6_if_state_change_oid));

/* Update the notification OID length. */
snmp_notification->OID.oid_len =
    IPV6_IF_STATE_CHANGE_OID_LEN;

/* Set first bound object as 'ipv6IfDescr'. */
memcpy (snmp_notification->snmp_object_list[0].Id,
        ipv6_if_descr_oid,
        sizeof(ipv6_if_descr_oid));

/* Set interface index in OID. */
snmp_notification->snmp_object_list[0].
    Id[(IPV6_IF_DESCR_OID_LEN - 1)] = 1;

/* Set OID length. */
snmp_notification->snmp_object_list[0].IdLen =
    IPV6_IF_DESCR_OID_LEN;

/* Set second bound object as 'ipv6IfOperStatus'. */
memcpy(snm_notification->snmp_object_list[1].Id,
        ipv6_if_oper_status_oid,
        sizeof(ipv6_if_oper_status_oid));

/* Update the interface index in the OID. */
snmp_notification->snmp_object_list[1].
    Id[(IPV6_IF_OPER_STATUS_OID_LEN - 1)] = 1;

/* Set OID length. */
snmp_notification->snmp_object_list[1].IdLen =
    IPV6_IF_OPER_STATUS_OID_LEN;

/* Update the number of bound objects. */
snmp_notification->snmp_object_list_len = 2;

/* Send the notification. */
SNMP_Notification_Ready (snmp_notification);
}
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Get\\_Notification\\_Ptr](#)

[CBSM\\_COMMUNITY\\_STRUCT](#)

## AgentGetAuthenTraps

This function indicates whether the authentication traps are enabled or disabled.

### Usage

```
BOOL AgentGetAuthenTraps (VOID);
```

### Return Values

- **NU\_TRUE**  
Authentication Trap is enabled.
- **NU\_FALSE**  
Authentication Trap is disabled.

### Description

Nucleus SNMP generates a trap when there is an unauthenticated attempt made to Nucleus SNMP if the authentication trap is enabled. The authentication trap indicator is initialized to the configurable variable `authentrap_onoff` in *networking/snmp/snmp\_cfg.c*. Its default value is ON (NU\_TRUE).

### Example

```
/* If Authentication Traps are disabled */  
if (AgentGetAuthenTraps () == NU_FALSE)  
{  
    /* Enable Authentication traps. */  
    AgentSetAuthenTraps (NU_TRUE);  
}
```

### Related Topics

[SNMP API Functions](#)

[AgentSetAuthenTraps](#)

## AgentSetAuthenTraps

This function enables/disables the authentication traps.

### Usage

```
VOID AgentSetAuthenTraps (BOOL enable);
```

### Arguments

- enable

Set this value to NU\_TRUE to enable the transmission of Authentication Traps. Set this value to NU\_FALSE to disable the transmission of Authentication Traps.

### Description

Nucleus SNMP generates a trap when there is an unauthenticated attempt made to Nucleus SNMP if the authentication trap is enabled. The authentication trap indicator is initialized to the configurable variable `authentrap_onoff` in *networking/snmp/snmp\_cfg.c*. Its default value is ON (NU\_TRUE).

### Example

```
/* If Authentication Traps are disabled */
if (AgentGetAuthenTraps () == NU_FALSE)
{
    /* Enable Authentication traps. */
    AgentSetAuthenTraps (NU_TRUE);
}
```

### Related Topics

[SNMP API Functions](#)

[AgentGetAuthenTraps](#)

## AgentSetColdTraps

This function enables/disables ColdStart Traps.

### Usage

```
VOID AgentSetColdTraps (BOOL enable);
```

### Arguments

- enable

Set this value to NU\_TRUE to enable the transmission of ColdStart Traps. Set this value to NU\_FALSE to disable the transmission of ColdStart Traps.

### Description

A ColdStart Trap signifies that the SNMP entity, supporting a notification originator application, is reinitializing itself and that its configuration may have been altered. ColdStart Trap indicator is initialized to the configurable variable coldtrap\_onoff in *networking/snmp/snmp\_cfg.c*. Its default value is ON (NU\_TRUE).

### Example

```
/* Enable coldStart trap. */  
AgentSetColdTraps (NU_TRUE);
```

### Related Topics

[SNMP API Functions](#)

[AgentSetAuthenTraps](#)

## SNMP\_Get\_Engine\_ID

This function returns the SNMP Engine ID and its length.

### Note



The SNMP Engine ID contains the IP address of the physical interface. If there are multiple interfaces, there is an equivalent number of SNMP Engine IDs. This function returns the Engine ID of the interface on which the last SNMP request was received.

### Usage

```
STATUS SNMP_Get_Engine_ID (UINT8  *engine_id,  
                          UINT32 *engine_id_len);
```

### Arguments

- **engine\_id**  
A pointer to the memory in which to store the Engine ID.
- **engine\_id\_len**  
Pointer to the variable containing Engine ID length.

### Return Values

- **NU\_SUCCESS**  
Engine ID obtained successfully.
- **SNMP\_ERROR**  
Invalid arguments.

### Example

```
UINT8      engine_id[SNMP_SIZE_SMALLOBJECTID];  
UINT32     engine_id_len;  
  
SNMP_Get_Engine_ID (engine_id, &engine_id_len);
```

### Related Topics

[SNMP API Functions](#)

[SNMP\\_Get\\_Notification\\_Ptr](#)



## CBSM\_Add\_Community

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS CBSM_Add_Community (CBSM_COMMUNITY_STRUCT *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry added to the CBSM Community table.
- SNMP\_ERROR  
Invalid argument, or community is already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Community table. This function does not save the node to the file system. If the node needs to be added to the file system, [CBSM\\_Save\\_Community](#) must be invoked. An entry can also be added to the Community table at compile-time by adding an entry in [CBSM\\_Community\\_Table](#) in *networking/snmp/snmp\_cfg.c* and modifying the value of CBSM\_MAX\_COMMUNITIES in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the CBSM Community Table.

---

**Note**

The *snmp\_api.h* file needs to be included.

---

```
CBSM_COMMUNITY_STRUCT    community;  
CBSM_COMMUNITY_STRUCT    *location_ptr;  
UINT8                    engine_id[SNMP_SIZE_SMALLOBJECTID];  
UINT32                    engine_id_len;  
UINT16                    status;  
  
/* Clear the structure. */  
memset (&community, 0, sizeof(CBSM_COMMUNITY_STRUCT));
```

```
/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (&engine_id, &engine_id_len);

/* Copy community index and index length. */
memcpy (community.cbsm_community_index, "comm1", 5);
community.cbsm_community_index_len = 5;

/* Copy community name. */
strcpy ((CHAR*) community.cbsm_community_name, "comm1");

/* Copy snmp Engine ID and ID length. */
memcpy (community.cbsm_engine_id, (VOID *) &engine_id,
        (unsigned int) engine_id_len);
community.cbsm_engine_id_len = (UINT8) engine_id_len;

/* Copy cbsm security name. */
strcpy((CHAR*)community.cbsm_security_name, CBSM_SECURITY_NAME);

#if (INCLUDE_MIB_CBSM == NU_TRUE)
/* Copy cbsm status */
community.cbsm_status = SNMP_ROW_ACTIVE;

/* Copy cbsm storage type. */
community.cbsm_storage_type = SNMP_STORAGE_NONVOLATILE;
#endif

/* Copy cbsm transport tag and tag length. */
memcpy (community.cbsm_transport_tag, CBSM_GROUP_NAME,
        CBSM_GROUP_NAME_LEN);
community.cbsm_transport_tag_len = CBSM_GROUP_NAME_LEN;

/* Copy cbsm context name. */
strcpy ((CHAR *) community.cbsm_context_name, "");

/* Add the community. */
CBSM_Add_Community (&community);

/* Find the community table entry. */
location_ptr = CBSM_Find_Community_Index ("comm1", NU_TRUE,
                                           &status);

/* Save the community entry in file, if enabled. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    CBSM_Save_Community (location_ptr);
#endif

/* Remove the community. */
CBSM_Remove_Community (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[CBSM\\_Save\\_Community](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## CBSM\_Find\_Community\_Index

This function returns a pointer to the community associated with the specified community index or the next community after the community associated with the specified community index. NU\_NULL is returned if there is no such community.

### Usage

```
CBSM_COMMUNITY_STRUCT *CBSM_Find_Community_Index(UINT8  *community_index,
                                                    UINT8  getflag,
                                                    UINT16 *status);
```

### Arguments

- **community\_index**  
Pointer to the community index being searched.
- **getflag**  
NU\_TRUE to return the community associated with the specified community index.  
NU\_FALSE to return the community after the community associated with the specified community index.
- **status**  
On returning from the function this variable will have a value of NU\_SUCCESS if a community was found, SNMP\_ERROR if no such community exists and SNMP\_GENERROR if the parameters were invalid.

### Return Values

- **CBSM\_COMMUNITY\_STRUCT \***  
When successful.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- **NU\_NULL**  
It is NU\_NULL if the getflag is NU\_TRUE and a matching community exists, or if getflag is NU\_FALSE and there are no more communities in the table.

### Example

The following is an example for adding, removing, finding and saving entries to the CBSM Community Table.

---

**Note**

The *snmp\_api.h* file needs to be included.

---

```
CBSM_COMMUNITY_STRUCT  community;
CBSM_COMMUNITY_STRUCT  *location_ptr;
UINT8                   engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                   engine_id_len;
```

```
UINT16                                status;

/* Clear the structure. */
memset (&community, 0, sizeof(CBSM_COMMUNITY_STRUCT));

/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (&engine_id, &engine_id_len);

/* Copy community index and index length. */
memcpy (community.cbsm_community_index, "comm1", 5);
community.cbsm_community_index_len = 5;

/* Copy community name. */
strcpy ((CHAR*) community.cbsm_community_name, "comm1");

/* Copy snmp engine id and id length. */
memcpy (community.cbsm_engine_id, (VOID *) &engine_id,
        (unsigned int) engine_id_len);
community.cbsm_engine_id_len = (UINT8) engine_id_len;

/* Copy cbsm security name. */
strcpy((CHAR*)community.cbsm_security_name, CBSM_SECURITY_NAME);

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Copy the cbsm status. */
    community.cbsm_status = SNMP_ROW_ACTIVE;

    /* Copy cbsm storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_NONVOLATILE;
#endif

/* Copy cbsm transport tag and tag length. */
memcpy (community.cbsm_transport_tag, CBSM_GROUP_NAME,
        CBSM_GROUP_NAME_LEN);
community.cbsm_transport_tag_len = CBSM_GROUP_NAME_LEN;

/* Copy cbsm context name. */
strcpy ((CHAR *) community.cbsm_context_name, "");

/* Add community. */
CBSM_Add_Community (&community);

/* Find Community table entry. */
location_ptr = CBSM_Find_Community_Index ("comm1", NU_TRUE,
                                           &status);

/* Save community entry in file, if enabled. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    CBSM_Save_Community (location_ptr);
#endif

/* Remove community. */
CBSM_Remove_Community (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[CBSM\\_Remove\\_Community](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## CBSM\_Remove\_Community

This function removes an entry from the community table. The entry is also removed from the file system if it was previously stored there. The memory previously allocated for this entry is also deallocated.

### Usage

```
STATUS CBSM_Remove_Community (CBSM_COMMUNITY_STRUCT *community);
```

### Arguments

- community

Pointer to the entry to be removed.

Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

If community is removed successfully.

- SNMP\_ERROR

If parameter passed is NU\_NULL, or community is not registered.

### Example

The following is an example for adding, removing, finding and saving entries to the CBSM Community Table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
CBSM_COMMUNITY_STRUCT    community;
CBSM_COMMUNITY_STRUCT    *location_ptr;
UINT8                    engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                   engine_id_len;
UINT16                   status;

/* Clear the structure. */
memset (&community, 0, sizeof(CBSM_COMMUNITY_STRUCT));

/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (&engine_id, &engine_id_len);

/* Copy community index and index length. */
memcpy (community.cbsm_community_index, "comm1", 5);
community.cbsm_community_index_len = 5;

/* Copy community name. */
strcpy ((CHAR*) community.cbsm_community_name, "comm1");
```

```
/* Copy snmp engine id and id length. */
memcpy (community.cbsm_engine_id, (VOID *) &engine_id,
        (unsigned int) engine_id_len);
community.cbsm_engine_id_len = (UINT8) engine_id_len;

/* Copy cbsm security name. */
strcpy((CHAR*)community.cbsm_security_name, CBSM_SECURITY_NAME);

#if (INCLUDE_MIB_CBSM == NU_TRUE)
/* Copy cbsm status. */
community.cbsm_status = SNMP_ROW_ACTIVE;

/* Copy cbsm storage type. */
community.cbsm_storage_type = SNMP_STORAGE_NONVOLATILE;
#endif

/* Copy cbsm transport tag and tag length. */
memcpy (community.cbsm_transport_tag, CBSM_GROUP_NAME,
        CBSM_GROUP_NAME_LEN);
community.cbsm_transport_tag_len = CBSM_GROUP_NAME_LEN;

/* Copy cbsm context name. */
strcpy ((CHAR *) community.cbsm_context_name, "");

/* Add community. */
CBSM_Add_Community (&community);

/* Find Community table entry. */
location_ptr = CBSM_Find_Community_Index ("comm1", NU_TRUE,
                                           &status);

/* Save community entry in file if enabled. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    CBSM_Save_Community (location_ptr);
#endif

/* Remove community. */
CBSM_Remove_Community (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[CBSM\\_Find\\_Community\\_Index](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## CBSM\_Save\_Community

This function saves an entry from the Community table to the file system.

### Note



SNMP must be configured to enable storage of data to the file system.

---

## Usage

```
STATUS CBSM_Save_Community (CBSM_COMMUNITY_STRUCT *community);
```

## Arguments

- **community**  
Pointer to the entry to be saved. Values for this entry are updated in the file system.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

## Return Values

- **NU\_SUCCESS**  
CBSM Community saved successfully.

## Description

This function needs to be invoked when a new node has been added, and the node needs to be saved to a file. This function also needs to be called after the values of an entry have been updated.

## Example

The following is an example for adding, removing, finding and saving entries to the CBSM Community Table.

### Note



The *snmp\_api.h* file needs to be included.

---

```
CBSM_COMMUNITY_STRUCT    community;  
CBSM_COMMUNITY_STRUCT    *location_ptr;  
UINT8                    engine_id[SNMP_SIZE_SMALLOBJECTID];  
UINT32                   engine_id_len;  
UINT16                   status;  
  
/* Clear the structure. */  
memset (&community, 0, sizeof(CBSM_COMMUNITY_STRUCT));  
  
/* Get SNMP Engine ID. */  
SNMP_Get_Engine_ID (&engine_id, &engine_id_len);  
  
/* Copy community index and index length. */  
memcpy (community.cbsm_community_index, "comm1", 5);
```



```
community.cbsm_community_index_len = 5;

/* Copy community name. */
strcpy ((CHAR*) community.cbsm_community_name, "comm1");

/* Copy snmp engine ID and ID length. */
memcpy (community.cbsm_engine_id, (VOID *) &engine_id,
        (unsigned int) engine_id_len);
community.cbsm_engine_id_len = (UINT8) engine_id_len;

/* Copy cbsm security name. */
strcpy((CHAR*)community.cbsm_security_name, CBSM_SECURITY_NAME);

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Copy cbsm status. */
    community.cbsm_status = SNMP_ROW_ACTIVE;

    /* Copy cbsm storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_NONVOLATILE;
#endif

/* Copy cbsm transport tag and tag length. */
memcpy (community.cbsm_transport_tag, CBSM_GROUP_NAME,
        CBSM_GROUP_NAME_LEN);
community.cbsm_transport_tag_len = CBSM_GROUP_NAME_LEN;

/* Copy cbsm context name. */
strcpy ((CHAR *) community.cbsm_context_name, "");

/* Add community. */
CBSM_Add_Community (&community);

/* Find Community table entry. */
location_ptr = CBSM_Find_Community_Index ("comm1", NU_TRUE,
                                           &status);

/* Save community entry in file, if enabled. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    CBSM_Save_Community (location_ptr);
#endif

/* Remove community. */
CBSM_Remove_Community (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[CBSM\\_Add\\_Community](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## SNMP\_Add\_To\_Notify\_Tbl

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS SNMP_Add_To_Notify_Tbl (SNMP_NOTIFY_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry added to Notify table.
- SNMP\_ERROR  
Invalid argument, or the Notify table entry is already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Notify Table. An entry can also be added to Notify table at compile-time by adding an entry in `Snm_Cfg_Notify_Tbl` in *networking/snmp/snmp\_cfg.c* and modifying the value of `NOTIFY_TBL_SIZE` in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_TABLE    notify_node;  
SNMP_NOTIFY_TABLE    *location_ptr;  
  
/* Initialize the Notify table.  */  
/* Clear the memory.  */  
memset (&notify_node, 0, sizeof(SNMP_NOTIFY_TABLE));  
  
/* Type */  
notify_node.snmp_notify_type = TRAP;
```

```
#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type. */
notify_node.snmp_notify_storage_type = SNMP_STORAGE_READONLY;

/* Row Status */
notify_node.snmp_notify_row_status = SNMP_ROW_ACTIVE;
#endif

/* Notify Name */
strcpy ((CHAR *) notify_node.snmp_notify_name, "notify1");

/* Tag Length */
notify_node.tag_len = 6;

/* Target Tag Name */
memcpy((VOID *)notify_node.snmp_notify_tag, (VOID *)"group1", 6);

/* Add entry */
SNMP_Add_To_Notify_Tbl (&notify_node);

/* Find the entry from the Notify table. */
location_ptr = SNMP_Find_Notify_Entry ("notify1");

/* Remove the entry found. */
SNMP_Remove_From_Notify_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Find\\_Notify\\_Entry](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## SNMP\_Find\_Notify\_Entry

This function returns an SNMP Notify Table entry based on the passed notify name.

### Usage

```
SNMP_NOTIFY_TABLE* SNMP_Find_Notify_Entry (UINT8 *notify_name);
```

### Arguments

- `notify_name`  
Pointer to notify name being searched.

### Return Values

- `SNMP_NOTIFY_TABLE`  
A pointer to the Notify table entry associated with the name passed into the routine.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- `NU_NULL`  
No matching Notify table entry exists.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_TABLE      notify_node;
SNMP_NOTIFY_TABLE      *location_ptr;

/* Initialize the Notify table.  */
/* Clear the memory.  */
memset (&notify_node, 0, sizeof(SNMP_NOTIFY_TABLE));

/* Type */
notify_node.snmp_notify_type = TRAP;

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
notify_node.snmp_notify_storage_type = SNMP_STORAGE_READONLY;

/* Row Status */
notify_node.snmp_notify_row_status = SNMP_ROW_ACTIVE;
#endif

/* Notify Name */
strcpy ((CHAR *) notify_node.snmp_notify_name, "notify1");

/* Tag Length */
```

```
notify_node.tag_len = 6;

/* Target Tag Name */
memcpy((VOID *)notify_node.snmp_notify_tag, (VOID *)"group1", 6);

/* Add entry */
SNMP_Add_To_Notify_Tbl (&notify_node);

/* Find the entry from Notify table. */
location_ptr = SNMP_Find_Notify_Entry ("notify1");

/* Remove the entry found. */
SNMP_Remove_From_Notify_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Add\\_To\\_Notify\\_Tbl](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## SNMP\_Remove\_From\_Notify\_Table

This function removes an entry from the Notify Table. The memory previously allocated for this entry is also deallocated.

### Usage

```
STATUS SNMP_Remove_From_Notify_Table (SNMP_NOTIFY_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be removed.  
Data structure [SNMP\\_NOTIFY\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Notify table entry is removed successfully.
- SNMP\_ERROR  
Invalid argument, or Notify table entry does not exist.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_TABLE      notify_node;  
SNMP_NOTIFY_TABLE      *location_ptr;  
  
/* Initialize the Notify table.  */  
/* Clear the memory.  */  
memset (&notify_node, 0, sizeof(SNMP_NOTIFY_TABLE));  
  
/* Type */  
notify_node.snmp_notify_type = TRAP;  
  
#if (INCLUDE_MIB_NO == NU_TRUE)  
/* Storage Type */  
notify_node.snmp_notify_storage_type = SNMP_STORAGE_READONLY;  
  
/* Row Status */  
notify_node.snmp_notify_row_status = SNMP_ROW_ACTIVE;  
#endif  
  
/* Notify Name */  
strcpy ((CHAR *) notify_node.snmp_notify_name, "notify1");
```

```
/* Tag Length */
notify_node.tag_len = 6;

/* Target Tag Name */
memcpy((VOID *)notify_node.snmp_notify_tag, (VOID *)"group1", 6);

/* Add entry. */
SNMP_Add_To_Notify_Tbl (&notify_node);

/* Find the entry from Notify table. */
location_ptr = SNMP_Find_Notify_Entry ("notify1");

/* Remove the entry found. */
SNMP_Remove_From_Notify_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Add\\_To\\_Notify\\_Tbl](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_TABLE](#)

## SNMP\_Add\_To\_Profile\_Tbl

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS SNMP_Add_To_Profile_Tbl (SNMP_NOTIFY_FILTER_PROFILE_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry added to the Notify Filter Profile table.
- SNMP\_ERROR  
Invalid argument, or the Notify Filter Profile table entry is already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Notify Filter Profile table. An entry can also be added to the Notify Filter Profile table at compile-time by adding an entry in `Snmp_Cfg_Fltr_Prof_Tbl` in *networking/snmp/snmp\_cfg.c* and modifying the value of `FLTR_PROF_TBL_SIZE` in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter Profile table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_PROFILE_TABLE    profile_node;  
SNMP_NOTIFY_FILTER_PROFILE_TABLE    *location_ptr;  
  
/* Initialize the Notify Filter Profile table entry. */  
/* Clear the memory. */  
memset (&profile_node, 0,  
        sizeof(SNMP_NOTIFY_FILTER_PROFILE_TABLE));  
  
#if (INCLUDE_MIB_NO == NU_TRUE)  
/* Storage Type */
```



```
profile_node.snmp_notify_filter_profile_storType =
    SNMP_STORAGE_READONLY;

/* Row Status */
profile_node.snmp_notify_filter_profile_row_status = SNMP_ROW_ACTIVE;
#endif

/* Params Name */
strcpy ((CHAR *) profile_node.snmp_target_params_name,
        "filterprof1");

/* Notify Filter Profile Name. */
strcpy ((CHAR *) profile_node.snmp_notify_filter_profile_name,
        "filterprof1");

/* Add entry. */
SNMP_Add_To_Profile_Tbl (&profile_node);

/* Find entry. */
location_ptr = SNMP_Find_Profile_Entry ("filterprof1");

/* Remove entry. */
SNMP_Remove_From_Profile_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Remove\\_From\\_Profile\\_Table](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#)

## SNMP\_Find\_Profile\_Entry

This function returns an entry from the Notify Filter Profile table based on the passed params name.

### Usage

```
SNMP_NOTIFY_FILTER_PROFILE_TABLE* SNMP_Find_Profile_Entry (
                                                    UINT8 *params_name);
```

### Arguments

- params\_name  
Pointer to params name being searched.

### Return Values

- SNMP\_NOTIFY\_FILTER\_PROFILE\_TABLE\*  
The Notify Filter Profile table entry was found.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- NU\_NULL  
The Notify Filter Profile table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter Profile table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_PROFILE_TABLE    profile_node;
SNMP_NOTIFY_FILTER_PROFILE_TABLE    *location_ptr;

/* Initialize the Notify Filter Profile table entry.  */
/* Clear the memory.  */
memset (&profile_node, 0,
        sizeof(SNMP_NOTIFY_FILTER_PROFILE_TABLE));

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
profile_node.snmp_notify_filter_profile_storType =
    SNMP_STORAGE_READONLY;

/* Row Status */
profile_node.snmp_notify_filter_profile_row_status =
    SNMP_ROW_ACTIVE;
#endif

/* Params Name */
```

```
strcpy ((CHAR *) profile_node.snmp_target_params_name,  
        "filterprof1");  
  
/* Notify Filter Profile Name */  
strcpy ((CHAR *) profile_node.snmp_notify_filter_profile_name,  
        "filterprof1");  
  
/* Add entry. */  
SNMP_Add_To_Profile_Tbl (&profile_node);  
  
/* Find entry. */  
location_ptr = SNMP_Find_Profile_Entry ("filterprof1");  
  
/* Remove entry. */  
SNMP_Remove_From_Profile_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Add\\_To\\_Profile\\_Tbl](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#)

## SNMP\_Remove\_From\_Profile\_Table

This function removes an entry from the Notify Filter Profile table. The memory previously allocated for this entry is also be deallocated.

### Usage

```
STATUS SNMP_Remove_From_Profile_Table (
    SNMP_NOTIFY_FILTER_PROFILE_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be removed.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The Notify Filter Profile table entry was removed successfully.
- SNMP\_ERROR  
Invalid argument, or an entry was not found in the Notify Filter Profile table.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter Profile table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_PROFILE_TABLE    profile_node;
SNMP_NOTIFY_FILTER_PROFILE_TABLE    *location_ptr;

/* Initialize the Notify Filter Profile table entry. */
/* Clear the memory */
memset (&profile_node, 0,
        sizeof(SNMP_NOTIFY_FILTER_PROFILE_TABLE));

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
profile_node.snmp_notify_filter_profile_storType =
    SNMP_STORAGE_READONLY;

/* Row Status */
profile_node.snmp_notify_filter_profile_row_status =
    SNMP_ROW_ACTIVE;
#endif

/* Params Name */
```

```
strcpy ((CHAR *) profile_node.snmp_target_params_name,  
        "filterprof1");  
  
/* Notify Filter Profile Name. */  
strcpy ((CHAR *) profile_node.snmp_notify_filter_profile_name,  
        "filterprof1");  
  
/* Add entry. */  
SNMP_Add_To_Profile_Tbl (&profile_node);  
  
/* Find entry. */  
location_ptr = SNMP_Find_Profile_Entry ("filterprof1");  
  
/* Remove entry. */  
SNMP_Remove_From_Profile_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Add\\_To\\_Profile\\_Tbl](#)

[SNMP Data Structures](#)

[SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#)

## SNMP\_Add\_To\_Filter\_Tbl

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS SNMP_Add_To_Filter_Tbl (SNMP_NOTIFY_FILTER_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added to the Notify Filter table.
- SNMP\_ERROR  
Invalid argument, or the Notify Filter table entry was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Notify Filter table. An entry can also be added to Notify Filter at compile-time by adding an entry in `Snm_Cfg_Fltr_Tbl` in *networking/snmp/snmp\_cfg.c* and modifying the value of `FLTR_TBL_SIZE` in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_TABLE      filter_node;
SNMP_NOTIFY_FILTER_TABLE      *location_ptr;
UINT32      subtree[6] = {1, 3, 6, 1, 1, 1};
UINT8      filter_mask[1] = {0xFC};
UINT8      maskLength = 1;

/* Initialize the Notify Filter table entry. */
/* Clear the memory. */
memset (&filter_node, 0, sizeof(SNMP_NOTIFY_FILTER_TABLE));
```

```
/* Filter Type */
filter_node.snmp_notify_filter_type = INCLUDED;

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
filter_node.snmp_notify_filter_storage_type =
    SNMP_STORAGE_READONLY;

/* Row Status */
filter_node.snmp_notify_filter_row_status = SNMP_ROW_ACTIVE;
#endif

/* Filter Name */
strcpy ((CHAR *) filter_node.snmp_notify_filter_profile_name,
    "filter1");

/* Subtree Length */
filter_node.subtree_len = 6;

/* Subtree */
memcpy (filter_node.snmp_notify_filter_subtree,
    subtree, 6 * sizeof(UINT32));

/* Mask Length (in number of bytes) */
filter_node.mask_len = (UINT32) maskLength;

/* Filter Mask */
memcpy (filter_node.snmp_notify_filter_mask,
    filter_mask, maskLength);

/* Add the entry to the list. */
SNMP_Add_To_Filter_Tbl (&filter_node);

/* Find entry. */
location_ptr = SNMP_Find_Filter_Entry ("filter1");

/* Remove entry. */
SNMP_Remove_From_Filter_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Find\\_Filter\\_Entry](#)

[SNMP\\_NOTIFY\\_FILTER\\_TABLE](#)

## SNMP\_Find\_Filter\_Entry

This function returns an entry from the Notify Filter table based on the passed filter name.

### Usage

```
SNMP_NOTIFY_FILTER_TABLE* SNMP_Find_Filter_Entry (UINT8 *filter_name);
```

### Arguments

- **filter\_name**  
Pointer to the filter name being searched.

### Return Values

- **SNMP\_NOTIFY\_FILTER\_TABLE**  
Entry in the Notify Filter table found with filter name passed in.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- **NU\_NULL**  
No entry in the Notify Filter table found with filter name passed in.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_TABLE      filter_node;
SNMP_NOTIFY_FILTER_TABLE      *location_ptr;
UINT32      subtree[6] = {1, 3, 6, 1, 1, 1};
UINT8      filter_mask[1] = {0xFC};
UINT8      maskLength = 1;

/* Initialize the Notify Filter table entry. */
/* Clear the memory. */
memset (&filter_node, 0, sizeof(SNMP_NOTIFY_FILTER_TABLE));

/* Filter Type */
filter_node.snmp_notify_filter_type = INCLUDED;

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
filter_node.snmp_notify_filter_storage_type =
    SNMP_STORAGE_READONLY;

/* Row Status */
filter_node.snmp_notify_filter_row_status = SNMP_ROW_ACTIVE;
#endif
```



```
/* Filter Name */
strcpy ((CHAR *) filter_node.snmp_notify_filter_profile_name,
        "filter1");

/* Subtree Length */
filter_node.subtree_len = 6;

/* Subtree */
memcpy (filter_node.snmp_notify_filter_subtree,
        subtree, 6 * sizeof(UINT32));

/* Mask Length (in number of bytes) */
filter_node.mask_len = (UINT32) maskLength;

/* Filter Mask */
memcpy (filter_node.snmp_notify_filter_mask,
        filter_mask, maskLength);

/* Add the entry to the list. */
SNMP_Add_To_Filter_Tbl (&filter_node);

/* Find entry. */
location_ptr = SNMP_Find_Filter_Entry ("filter1");

/* Remove entry. */
SNMP_Remove_From_Filter_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Remove\\_From\\_Filter\\_Table](#)

[SNMP\\_NOTIFY\\_FILTER\\_TABLE](#)

## SNMP\_Remove\_From\_Filter\_Table

This function removes an entry from the Notify Filter table. The memory previously allocated for this entry is also be deallocated.

### Usage

```
STATUS SNMP_Remove_From_Filter_Table (SNMP_NOTIFY_FILTER_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be removed.  
Data structure [SNMP\\_NOTIFY\\_FILTER\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
An entry from the Notify Filter table was removed successfully.
- SNMP\_ERROR  
Invalid argument, or the entry was not found in the Notify Filter table.

### Example

The following is an example for adding, removing, finding and saving entries to the Notify Filter table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_NOTIFY_FILTER_TABLE filter_node;
SNMP_NOTIFY_FILTER_TABLE *location_ptr;
UINT32 subtree[6] = {1, 3, 6, 1, 1, 1};
UINT8 filter_mask[1] = {0xFC};
UINT8 maskLength = 1;

/* Initialize the Notify Filter table entry. */
/* Clear the memory. */
memset (&filter_node, 0, sizeof(SNMP_NOTIFY_FILTER_TABLE));

/* Filter Type */
filter_node.snmp_notify_filter_type = INCLUDED;

#if (INCLUDE_MIB_NO == NU_TRUE)
/* Storage Type */
filter_node.snmp_notify_filter_storage_type =
    SNMP_STORAGE_READONLY;

/* Row Status */
filter_node.snmp_notify_filter_row_status = SNMP_ROW_ACTIVE;
```

```
#endif

/* Filter Name */
strcpy ((CHAR *) filter_node.snmp_notify_filter_profile_name,
        "filter1");

/* Subtree Length */
filter_node.subtree_len = 6;

/* Subtree */
memcpy (filter_node.snmp_notify_filter_subtree,
        subtree, 6 * sizeof(UINT32));

/* Mask Length (in number of bytes) */
filter_node.mask_len = (UINT32) maskLength;

/* Filter Mask */
memcpy (filter_node.snmp_notify_filter_mask,
        filter_mask, maskLength);

/* Add the entry to the list. */
SNMP_Add_To_Filter_Tbl (&filter_node);

/* Find entry. */
location_ptr = SNMP_Find_Filter_Entry ("filter1");

/* Remove entry. */
SNMP_Remove_From_Filter_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Find\\_Filter\\_Entry](#)

[SNMP\\_NOTIFY\\_FILTER\\_TABLE](#)

## SNMP\_Add\_Target

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS SNMP_Add_Target (SNMP_TARGET_ADDRESS_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added successfully to the Target Address table.
- SNMP\_ERROR  
Invalid argument.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Target Address table. An entry can also be added to the Target Address table at compile-time by adding an entry in `Snm_Cfg_Tgr_Addr_Tbl` in *networking/snmp/snmp\_cfg.c* and modifying the value of `TGR_ADDR_TBL_SIZE` in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Address table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_ADDRESS_TABLE    tgr_node;  
SNMP_TARGET_ADDRESS_TABLE    *location_ptr;  
UINT8                        address[4] = {192, 168, 0, 1};  
  
/* Clear the memory. */  
memset (&tgr_node, 0, sizeof(SNMP_TARGET_ADDRESS_TABLE));  
  
/* Host name */  
strcpy ((CHAR *) tgr_node.snmp_target_addr_name, "hosttemp1");
```

```
/* IP Address */
memcpy (tgr_node.snmp_target_addr_tAddress, address, 4);

/* IP Address Type */
tgr_node.snmp_target_addr_tfamily = NU_FAMILY_IP;

/* Parameters */
tgr_node.params_len = 15;
memcpy (tgr_node.snmp_target_addr_params,
        "NoAuthNoPriv-v1", 15);

#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Storage Type */
tgr_node.snmp_target_addr_storage_type = SNMP_STORAGE_READONLY;

/* Row Status */
tgr_node.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Domain */
tgr_node.snmp_target_addr_tDomain = SNMP_UDP;

/* Tag list length */
tgr_node.tag_list_len = 6;

/* Tag list */
memcpy (tgr_node.snmp_target_addr_tag_list,
        "group1", 6);

/* Add the entry. */
SNMP_Add_Target (&tgr_node);

/* Find the entry. */
location_ptr = SNMP_Find_Target ("hosttempl");

/* Remove the entry. */
SNMP_Remove_From_Tgr_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Find\\_Target](#)

[SNMP\\_SS\\_STRUCT](#)

## SNMP\_Find\_Target

This function finds an entry in the Target Address table and returns a pointer to it. If the entry is not found, NU\_NULL is returned.

### Usage

```
SNMP_TARGET_ADDRESS_TABLE* SNMP_Find_Target (UINT8 *target_name);
```

### Arguments

- **target\_name**  
Pointer to the name of the entry to be found.

### Return Values

- [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#)  
A pointer to the Target Address table entry found with specified target\_name.  
Data structure [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- **NU\_NULL**  
No Target Address table entry found with the specified target\_name.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Address table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_ADDRESS_TABLE    tgr_node;  
SNMP_TARGET_ADDRESS_TABLE    *location_ptr;  
UINT8                        address[4] = {192, 168, 0, 1};  
  
/* Clear the memory. */  
memset (&tgr_node, 0, sizeof(SNMP_TARGET_ADDRESS_TABLE));  
  
/* Host name */  
strcpy ((CHAR *) tgr_node.snmp_target_addr_name, "hosttempl");  
  
/* IP Address */  
memcpy (tgr_node.snmp_target_addr_tAddress, address, 4);  
  
/* IP Address Type */  
tgr_node.snmp_target_addr_tfamily = NU_FAMILY_IP;  
  
/* Parameters */  
tgr_node.params_len = 15;  
memcpy (tgr_node.snmp_target_addr_params,  
        "NoAuthNoPriv-v1", 15);
```

```
#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Storage Type */
tgr_node.snmp_target_addr_storage_type = SNMP_STORAGE_READONLY;

/* Row Status */
tgr_node.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Domain */
tgr_node.snmp_target_addr_tDomain = SNMP_UDP;

/* Tag list length */
tgr_node.tag_list_len = 6;

/* Tag list */
memcpy (tgr_node.snmp_target_addr_tag_list,
        "group1", 6);

/* Add the entry. */
SNMP_Add_Target (&tgr_node);

/* Find the entry. */
location_ptr = SNMP_Find_Target ("hosttemp1");

/* Remove the entry. */
SNMP_Remove_From_Tgr_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Add\\_Target](#)

[SNMP\\_SS\\_STRUCT](#)

## SNMP\_Remove\_From\_Tgr\_Table

This function removes an entry from the Target Address table. The memory previously allocated for this entry is also deallocated.

### Usage

```
STATUS SNMP_Remove_From_Tgr_Table (SNMP_TARGET_ADDRESS_TABLE *node);
```

### Arguments

- **node**  
Pointer to the entry to be removed.  
Data structure [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
The Target Address table entry was removed successfully.
- **SNMP\_ERROR**  
Invalid argument, or the entry in the Target Address table did not exist.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Address table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_ADDRESS_TABLE    tgr_node;  
SNMP_TARGET_ADDRESS_TABLE    *location_ptr;  
UINT8                        address[4] = {192, 168, 0, 1};  
  
/* Clear the memory. */  
memset (&tgr_node, 0, sizeof(SNMP_TARGET_ADDRESS_TABLE));  
  
/* Host name */  
strcpy ((CHAR *) tgr_node.snmp_target_addr_name, "hosttempl");  
  
/* IP Address */  
memcpy (tgr_node.snmp_target_addr_tAddress, address, 4);  
  
/* IP Address Type */  
tgr_node.snmp_target_addr_tfamily = NU_FAMILY_IP;  
  
/* Parameters */  
tgr_node.params_len = 15;  
memcpy (tgr_node.snmp_target_addr_params,  
        "NoAuthNoPriv-v1", 15);
```



```
#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Storage Type. */
tgr_node.snmp_target_addr_storage_type = SNMP_STORAGE_READONLY;

/* Row Status */
tgr_node.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Domain */
tgr_node.snmp_target_addr_tDomain = SNMP_UDP;

/* Tag list length */
tgr_node.tag_list_len = 6;

/* Tag list */
memcpy (tgr_node.snmp_target_addr_tag_list,
        "group1", 6);

/* Add the entry. */
SNMP_Add_Target (&tgr_node);

/* Find the entry. */
location_ptr = SNMP_Find_Target ("hosttemp1");

/* Remove the entry. */
SNMP_Remove_From_Tgr_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Find\\_Target](#)

[SNMP\\_SS\\_STRUCT](#)

## SNMP\_Add\_Params

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS SNMP_Add_Params (SNMP_TARGET_PARAMS_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [SNMP\\_TARGET\\_PARAMS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added to the Target Params table.
- SNMP\_ERROR  
Invalid argument, or the Target Params table entry was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Target Params table. An entry can also be added to the Target Params table at compile-time by adding an entry in `Snm_Cfg_Tgr_Params_Tbl` in *networking/snmp/snmp\_cfg.c* and modifying the value of `TGR_PARAMS_TBL_SIZE` in *include/networking/snmp\_cfg.h*.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Params table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_PARAMS_TABLE      param_node;
SNMP_TARGET_PARAMS_TABLE      *location_ptr;

/* Clear the memory. */
memset (&param_node, 0, sizeof(SNMP_TARGET_PARAMS_TABLE));

#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Storage Type */
param_node.snmp_target_params_storage_type =
    SNMP_STORAGE_READONLY;
```

```
    /* Row Status */
    param_node.snmp_target_params_row_status = SNMP_ROW_ACTIVE;
#endif

/* Target Params Name */
strcpy ((CHAR *) param_node.snmp_target_params_name, "params");

/* MP Model*/
param_node.snmp_target_params_mp_model = SNMP_VERSION_V3;

/* Security Model */
param_node.snmp_target_params_security_model = SNMP_USM;

/* Security Name */
strcpy ((CHAR *) param_node.snmp_target_params_security_name,
        "initial");

/* Security Level */
param_node.snmp_target_params_security_level =
        SNMP_SECURITY_AUTHNOPRIV;

/* Add entry. */
SNMP_Add_Params (&param_node);

/* Find entry. */
location_ptr = SNMP_Find_Params ("params");

/* Remove entry. */
SNMP_Remove_From_Params_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP\\_Find\\_Params](#)

[SNMP Data Structures](#)

[SNMP\\_TARGET\\_PARAMS\\_TABLE](#)

## SNMP\_Find\_Params

This function finds an entry in the Params table and returns a pointer to it. If the entry is not found, NU\_NULL is returned.

### Usage

```
SNMP_TARGET_PARAMS_TABLE* SNMP_Find_Params (UINT8 *params_name);
```

### Arguments

- `params_name`  
Pointer to the name of the entry to be found.

### Return Values

- [SNMP\\_TARGET\\_PARAMS\\_TABLE](#)  
Pointer to the Target Params entry associated with the name passed into the routine.  
Data structure [SNMP\\_TARGET\\_PARAMS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- `NU_NULL`  
A matching entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Params table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_PARAMS_TABLE      param_node;
SNMP_TARGET_PARAMS_TABLE      *location_ptr;

/* Clear the memory. */
memset (&param_node, 0, sizeof(SNMP_TARGET_PARAMS_TABLE));

#if (INCLUDE_MIB_TARGET == NU_TRUE)
    /* Storage Type. */
    param_node.snmp_target_params_storage_type = SNMP_STORAGE_READONLY;

    /* Row Status. */
    param_node.snmp_target_params_row_status = SNMP_ROW_ACTIVE;
#endif

/* Target Params Name. */
strcpy ((CHAR *) param_node.snmp_target_params_name, "params");

/* MP Model */
param_node.snmp_target_params_mp_model = SNMP_VERSION_V3;
```

```
/* Security Model */
param_node.snmp_target_params_security_model = SNMP_USM;

/* Security Name */
strcpy ((CHAR *) param_node.snmp_target_params_security_name,
        "initial");

/* Security Level */
param_node.snmp_target_params_security_level =
        SNMP_SECURITY_AUTHNOPRIV;

/* Add entry. */
SNMP_Add_Params (&param_node);

/* Find entry. */
location_ptr = SNMP_Find_Params ("params");

/* Remove entry. */
SNMP_Remove_From_Params_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Add\\_Params](#)

[SNMP\\_TARGET\\_PARAMS\\_TABLE](#)

## SNMP\_Remove\_From\_Params\_Table

This function removes an entry from the Target Params table. The memory previously allocated for this entry is also deallocated.

### Usage

```
STATUS SNMP_Remove_From_Params_Table (SNMP_TARGET_PARAMS_TABLE *node);
```

### Arguments

- node  
Pointer to the entry to be removed.  
Data structure [SNMP\\_TARGET\\_PARAMS\\_TABLE](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Target Params table entry was removed successfully.
- SNMP\_ERROR  
Invalid argument, or entry was not found in Target Params table.

### Example

The following is an example for adding, removing, finding and saving entries to the Target Params table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
SNMP_TARGET_PARAMS_TABLE      param_node;  
SNMP_TARGET_PARAMS_TABLE      *location_ptr;  
  
/* Clear the memory. */  
memset (&param_node, 0, sizeof(SNMP_TARGET_PARAMS_TABLE));  
  
#if (INCLUDE_MIB_TARGET == NU_TRUE)  
    /* Storage Type */  
    param_node.snmp_target_params_storage_type = SNMP_STORAGE_READONLY;  
  
    /* Row Status */  
    param_node.snmp_target_params_row_status = SNMP_ROW_ACTIVE;  
#endif  
  
/* Target Params Name */  
strcpy ((CHAR *) param_node.snmp_target_params_name, "params");  
  
/* MP Model*/  
param_node.snmp_target_params_mp_model = SNMP_VERSION_V3;
```

```
/* Security Model */
param_node.snmp_target_params_security_model = SNMP_USM;

/* Security Name */
strcpy ((CHAR *) param_node.snmp_target_params_security_name,
        "initial");

/* Security Level */
param_node.snmp_target_params_security_level =
    SNMP_SECURITY_AUTHNOPRIV;

/* Add entry. */
SNMP_Add_Params (&param_node);

/* Find entry. */
location_ptr = SNMP_Find_Params ("params");

/* Remove entry. */
SNMP_Remove_From_Params_Table (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[SNMP\\_Find\\_Params](#)

[SNMP\\_TARGET\\_PARAMS\\_TABLE](#)

## USM\_Add\_User

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS USM_Add_User (USM_USERS_STRUCT *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [USM\\_USERS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added to the USM User table.
- SNMP\_ERROR  
Invalid argument, or the user was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the USM (User-based Security Model) User table. This function does not save the node to the file system. If the node needs to be added to the file system, [USM\\_Save\\_User](#) must be invoked. An entry can also be added to the USM User table at compile-time by adding an entry in [Usm\\_User\\_Table](#) in *networking/snmp/snmp\_cfg.c* and modifying the value of USM\_MAX\_USER\_USERS in *include/networking/snmp\_cfg.h*. Refer to “[Advanced SNMP Topics](#)” on page 1693 for more information on USM.

### Example

The following is an example for adding, removing, finding and saving entries to the USM View User table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

USM_USERS_STRUCT	*location_ptr;
USM_USERS_STRUCT	user;
USM_AUTH_PROT_STRUCT	*auth_prot;
UINT8	template_pass[SNMP_SIZE_SMALLOBJECTID] =
	"templatepass";
UINT8	template_pass_len;
STATUS	status = NU_SUCCESS;
UINT8	engine_id[SNMP_SIZE_SMALLOBJECTID];



```
UINT32                engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case. */
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get SNMP Engine ID */
SNMP_Get_Engine_ID (engine_id, &engine_id_len);

/* Add Template for MD5. */
memcpy (user.usm_user_engine_id, engine_id,
        (unsigned int) engine_id_len);

/* Copy engine_id len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use the HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* Use the CBC-DES Privacy Protocol. */
user.usm_priv_index = USM_DES;

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Copy USM storage type. */
    user.usm_storage_type = SNMP_STORAGE_PERMANENT;

    /* Copy the USM row status. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Make the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, then it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
```

```
        user.usm_auth_key, auth_prot->key_length);

/* This is also the privacy key. */
memcpy (user.usm_priv_key, user.usm_auth_key,
        auth_prot->key_length);
memcpy (user.usm_own_priv_key, user.usm_auth_key,
        auth_prot->key_length);

/* Add this entry to the usm_user_table. */
status = USM_Add_User (&user);
}

/* Find the user by the security name. */
location_ptr = USM_Lookup_Users ("template");

#ifdef (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    USM_Save_User (location_ptr);
#endif

/* Remove the user. */
USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Lookup\\_Users](#)

[USM\\_USERS\\_STRUCT](#)

## USM\_Lookup\_Users

This function returns a pointer to a user entry based on the user security name passed in.

### Usage

```
USM_USERS_STRUCT* USM_Lookup_Users (UINT8 *msg_user_name);
```

### Arguments

- `msg_user_name`  
Pointer to the user security name being searched.

### Return Values

- [USM\\_USERS\\_STRUCT](#)\*  
The USM user was found with the specified security name (`msg_user_name`).  
Data structure [USM\\_USERS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- `NU_NULL`  
The USM user was not found with the specified security name.

### Example

The following is an example for adding, removing, finding and saving entries to the USM User table.

#### Note



The *snmp\_api.h* file needs to be included.

```
USM_USERS_STRUCT      *location_ptr;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;
UINT8                  template_pass[SNMP_SIZE_SMALLOBJECTID] =
                        "templatepass";
UINT8                  template_pass_len;
STATUS                 status = NU_SUCCESS;
UINT8                  engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                 engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case. */
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (engine_id, &engine_id_len);

/* Add the template for MD5. */
memcpy (user.usm_user_engine_id, engine_id,
        (unsigned int) engine_id_len);
```

```
/* Copy the engine_id_len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy the user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy the USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use the HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* Use the CBC-DES Privacy Protocol. */
user.usm_priv_index = USM_DES;

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Copy the USM storage type. */
    user.usm_storage_type = SNMP_STORAGE_PERMANENT;

    /* Copy the USM row status. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Make the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, then it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
            user.usm_auth_key, auth_prot->key_length);

    /* This is also the privacy key. */
    memcpy (user.usm_priv_key, user.usm_auth_key,
            auth_prot->key_length);
    memcpy (user.usm_own_priv_key, user.usm_auth_key,
            auth_prot->key_length);

    /* Add this entry to the usm_user_table. */
    status = USM_Add_User (&user);
}

/* Find user by security name. */
location_ptr = USM_Lookup_Users ("template");
```

```
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    USM_Save_User (location_ptr);
#endif

/* Remove the user. */
USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Add\\_User](#)

[USM\\_USERS\\_STRUCT](#)

## USM\_Remove\_User

This function removes an entry from the USM User table. The entry is also removed from the file system, if it was previously stored there. The memory previously allocated for this entry is also deallocated.

### Usage

```
STATUS USM_Remove_User (USM_USERS_STRUCT *user);
```

### Arguments

- **user**  
Pointer to the entry to be removed.  
Data structure [USM\\_USERS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
User removed successfully.
- **SNMP\_ERROR**  
Invalid argument or user not found.

### Example

The following is an example for adding, removing, finding and saving entries to the USM User table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
USM_USERS_STRUCT      *location_ptr;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;
UINT8                 template_pass[SNMP_SIZE_SMALLOBJECTID] =
                        "templatepass";
UINT8                 template_pass_len;
STATUS                 status = NU_SUCCESS;
UINT8                 engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                 engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case. */
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (engine_id, &engine_id_len);

/* Add Template for MD5. */
```

```
memcpy (user.usm_user_engine_id, engine_id,
        (unsigned int) engine_id_len);

/* Copy engine_id_len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use the HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* Use the CBC-DES Privacy Protocol. */
user.usm_priv_index = USM_DES;

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Copy USM storage type. */
    user.usm_storage_type = SNMP_STORAGE_PERMANENT;

    /* Copy USM row status. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Create the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
            user.usm_auth_key, auth_prot->key_length);

    /* This is also the privacy key. */
    memcpy (user.usm_priv_key, user.usm_auth_key,
            auth_prot->key_length);
    memcpy (user.usm_own_priv_key, user.usm_auth_key,
            auth_prot->key_length);

    /* Add this entry to the usm_user_table. */
    status = USM_Add_User (&user);
}
```

```
/* Find the user using the security name. */
location_ptr = USM_Lookup_Users ("template");

#ifdef (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    USM_Save_User (location_ptr);
#endif

/* Remove the user. */
USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Add\\_User](#)

[USM\\_USERS\\_STRUCT](#)



## USM\_Save\_User

This function saves an entry from the USM User to the file system.

---

**Note**

---



SNMP must be configured to enable storage of data to the file system.

---

### Usage

```
STATUS USM_Save_User (USM_USERS_STRUCT *user);
```

### Arguments

- **user**  
Pointer to the entry to be saved. Values for this entry are updated in the file system.  
Data structure [USM\\_USERS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
User saved successfully.

### Description

This function needs to be invoked when a new node has been added and the node needs to be saved to file. It also needs to be called after the values of an entry have been updated.

### Example

The following is an example for adding, removing, finding and saving entries to the USM User table.

---

**Note**

---



The *snmp\_api.h* file needs to be included.

---

```
USM_USERS_STRUCT      *location_ptr;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;
UINT8                 template_pass[SNMP_SIZE_SMALLOBJECTID] =
                        "templatepass";
UINT8                 template_pass_len;
STATUS                 status = NU_SUCCESS;
UINT8                 engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                 engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case*/
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get SNMP Engine ID. */
```

```

SNMP_Get_Engine_ID (engine_id, &engine_id_len);

/* Add the template for MD5. */
memcpy (user.usm_user_engine_id, engine_id,
        (unsigned int) engine_id_len);

/* Copy engine_id_len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* Use the CBC-DES Privacy Protocol. */
user.usm_priv_index = USM_DES;

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Copy the USM storage type. */
    user.usm_storage_type = SNMP_STORAGE_PERMANENT;

    /* Copy USM row status. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Create the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
            user.usm_auth_key, auth_prot->key_length);

    /* This is also the privacy key. */
    memcpy (user.usm_priv_key, user.usm_auth_key,
            auth_prot->key_length);
    memcpy (user.usm_own_priv_key, user.usm_auth_key,
            auth_prot->key_length);
}

```

```
    /* Add this entry to the usm_user_table. */
    status = USM_Add_User (&user);
}

/* Find the user using security name. */
location_ptr = USM_Lookup_Users ("template");

#ifdef (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    USM_Save_User (location_ptr);
#endif

/* Remove user. */
USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Remove\\_User](#)

[USM\\_USERS\\_STRUCT](#)

## USM\_Lookup\_Priv\_Prot

This function returns a pointer to an entry for the Privacy Protocol corresponding to the passed index.

### Usage

```
USM_PRIV_PROT_STRUCT* USM_Lookup_Priv_Prot (UINT32 index);
```

### Arguments

- index  
Index for the Privacy Protocol being searched.

### Return Values

- USM\_PRIV\_PROT\_STRUCT  
A pointer to the Privacy Protocol registered with Nucleus SNMP.  
Data structure [USM\\_PRIV\\_PROT\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- NU\_NULL  
Privacy Protocol is not registered with Nucleus SNMP.

### Example

The following is an example for adding, removing, finding and saving entries to the USM User table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
USM_USERS_STRUCT      *location_ptr;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;
UINT8                 template_pass[SNMP_SIZE_SMALLOBJECTID] =
                        "templatepass";
UINT8                 template_pass_len;
STATUS                status = NU_SUCCESS;
UINT8                 engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case*/
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get the SNMP Engine ID. */
SNMP_Get_Engine_ID (engine_id, &engine_id_len);

/* Add the template for MD5. */
memcpy (user.usm_user_engine_id, engine_id,
```

```
(unsigned int) engine_id_len);

/* Copy engine_id_len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy the user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy the USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use the HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* If CBC-DES Privacy Protocol is registered with Nucleus SNMP
 * then use it, otherwise don't use the privacy service.
 */
if (USM_Lookup_Priv_Prot(USM_DES) != NU_NULL)
    user.usm_priv_index = USM_DES;
else
    user.usm_priv_index = USM_NOPRIV;

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Copy the USM storage type. */
    user.usm_storage_type = SNMP_STORAGE_PERMANENT;

    /* Copy the USM row status. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Create the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
            user.usm_auth_key, auth_prot->key_length);

    /* This is also the privacy key. */
    memcpy (user.usm_priv_key, user.usm_auth_key,
            auth_prot->key_length);
    memcpy (user.usm_own_priv_key, user.usm_auth_key,
            auth_prot->key_length);
}
```

```
        /* Add this entry to the usm_user_table. */
        status = USM_Add_User (&user);
    }

    /* Find the user by security name. */
    location_ptr = USM_Lookup_Users ("template");

    #if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
        USM_Save_User (location_ptr);
    #endif

    /* Remove the user. */
    USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Lookup\\_Auth\\_Prot](#)

[USM\\_PRIV\\_PROT\\_STRUCT](#)

## USM\_Lookup\_Auth\_Prot

This function returns a pointer to the entry for the Authentication Protocol that corresponds to the passed index.

### Usage

```
USM_AUTH_PROT_STRUCT* USM_Lookup_Auth_Prot (UINT32 index);
```

### Arguments

- index  
Index for the Authentication Protocol being searched. Nucleus SNMP supports HMAC-MD5 and HMAC-SHA Authentication Protocols. The Authentication Protocol index for HMAC-MD5 is USM\_MD5, for HMAC-SHA is USM\_SHA and for No Authentication Protocol is USM\_NOAUTH.

### Return Values

- USM\_AUTH\_PROT\_STRUCT  
Pointer to the Authentication Protocol registered with Nucleus SNMP.  
Data structure [USM\\_AUTH\\_PROT\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.
- NU\_NULL  
The Authentication Protocol is not registered with Nucleus SNMP.

### Example

The following is an example for adding, removing, finding and saving entries to the USM User table.

#### Note



The *snmp\_api.h* file needs to be included.

```
USM_USERS_STRUCT      *location_ptr;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;
UINT8                  template_pass[SNMP_SIZE_SMALLOBJECTID] =
                        "templatepass";
UINT8                  template_pass_len;
STATUS                 status = NU_SUCCESS;
UINT8                  engine_id[SNMP_SIZE_SMALLOBJECTID];
UINT32                 engine_id_len;

template_pass_len = (UINT8) strlen ("templatepass");

/* The following two templates are added for any case. */
memset (&user, 0, sizeof(USM_USERS_STRUCT));

/* Get SNMP Engine ID. */
SNMP_Get_Engine_ID (engine_id, &engine_id_len);
```

```
/* Add the template for MD5. */
memcpy (user.usm_user_engine_id, engine_id,
        (unsigned int) engine_id_len);

/* Copy engine_id_len. */
user.usm_user_engine_id_len = (UINT8) engine_id_len;

/* Copy the user name. */
strcpy ((CHAR *) user.usm_user_name, "template");

/* Copy the USM security name. */
strcpy ((CHAR *) user.usm_security_name, "template");

/* Use the HMAC-MD5 Authentication Protocol. */
user.usm_auth_index = USM_MD5;

/* Use the CBC-DES Privacy Protocol. */
user.usm_priv_index = USM_DES;

#if (INCLUDE_MIB_USM == NU_TRUE)
/* Copy the USM storage type. */
user.usm_storage_type = SNMP_STORAGE_PERMANENT;

/* Copy the USM row status. */
user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Create the authentication key using the authentication password. */

/* Get the handle to the HMAC-MD5 Authentication Protocol. */
auth_prot = USM_Lookup_Auth_Prot (user.usm_auth_index);

/* If HMAC-MD5 is not registered with Nucleus SNMP, or it doesn't
 * have a handle to the routine that calculates the authentication
 * key using the password, it is an error condition.
 */
if ((auth_prot == NU_NULL) ||
    (auth_prot->usm_password_cb == NU_NULL))
{
    status = SNMP_ERROR;
}

if (status == NU_SUCCESS)
{
    /* Get the Authentication Key for HMAC-MD5. */
    auth_prot->usm_password_cb (template_pass, template_pass_len,
                                user.usm_auth_key);
    memcpy (user.usm_own_auth_key,
            user.usm_auth_key, auth_prot->key_length);

    /* This is also the privacy key. */
    memcpy (user.usm_priv_key, user.usm_auth_key,
            auth_prot->key_length);
    memcpy (user.usm_own_priv_key, user.usm_auth_key,
            auth_prot->key_length);

    /* Add this entry to the usm_user_table. */
}
```



```
        status = USM_Add_User (&user);
    }

    /* Find the user by security name. */
    location_ptr = USM_Lookup_Users ("template");

    #if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
        USM_Save_User (location_ptr);
    #endif

    /* Remove the user. */
    USM_Remove_User (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[USM\\_Lookup\\_Priv\\_Prot](#)

[USM\\_AUTH\\_PROT\\_STRUCT](#)

## VACM\_Add\_Context

This function adds an entry to the View-based Access Control Model (VACM) Context table. Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on the VACM Context Table.

### Usage

```
STATUS VACM_Add_Context (UINT8 *context_name);
```

### Arguments

- context\_name  
Pointer to the context name to be added.

### Return Values

- NU\_SUCCESS  
The context name was successfully added in VACM Context table.
- SNMP\_ERROR  
There is a NU\_NULL pointer for the specified context\_name, the context\_name is pointing to a string of length greater than 32, or the context\_name is already registered with Nucleus SNMP.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Context table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
VACM_CONTEXT_STRUCT    *location_ptr;

/* Add context. */
VACM_Add_Context ((UINT8 *) "context1");

/* Find context. */
VACM_Search_Context ("context1", &location_ptr);

/* Remove context. */
VACM_Remove_Context ((UINT8 *) "context1");
```

## Related Topics

[SNMP API Functions](#)

[VACM\\_Search\\_Context](#)

[VACM\\_CONTEXT\\_STRUCT](#)

## VACM\_Search\_Context

This function returns a pointer to the context corresponding to the passed context name.

### Usage

```
STATUS VACM_Search_Context (UINT8          *context_name,  
                           VACM_CONTEXT_STRUCT **loc_ptr);
```

### Arguments

- `context_name`  
Pointer to context name being searched.
- `loc_ptr`  
Double pointer to the Context entry.  
Data structure [VACM\\_CONTEXT\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
The context table entry was found.
- `SNMP_ERROR`  
The context table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Context table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
VACM_CONTEXT_STRUCT    *location_ptr;  
  
/* Add context. */  
VACM_Add_Context ((UINT8 *) "context1");  
  
/* Find context. */  
VACM_Search_Context ("context1", &location_ptr);  
  
/* Remove context. */  
VACM_Remove_Context ((UINT8 *) "context1");
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_Context](#)

[VACM\\_CONTEXT\\_STRUCT](#)

## VACM\_Remove\_Context

This function removes an entry from the Context Name table.

### Usage

```
STATUS VACM_Remove_Context (UINT8 *context_name);
```

### Arguments

- `context_name`  
Pointer to the context name to be removed.

### Return Values

- `NU_SUCCESS`  
Context table entry removed successfully.
- `SNMP_ERROR`  
Invalid arguments, or the Context Name table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Context table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
VACM_CONTEXT_STRUCT    *location_ptr;

/* Add context. */
VACM_Add_Context ((UINT8 *) "context1");

/* Find context. */
VACM_Search_Context ("context1", &location_ptr);

/* Remove context. */
VACM_Remove_Context ((UINT8 *) "context1");
```

### Related Topics

[SNMP API Functions](#)

[VACM\\_Search\\_Context](#)

[VACM\\_CONTEXT\\_STRUCT](#)

## VACM\_InsertGroup

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS VACM_InsertGroup (VACM_SEC2GROUP_STRUCT *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [VACM\\_SEC2GROUP\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The was entry added to Security to Group table.
- SNMP\_ERROR  
Invalid argument, or the Security to Group table entry was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the Security to Group table. This function does not save the node to the file system. If the node needs to be added to the file system, [VACM\\_Save\\_Group](#) must be invoked. An entry can also be added to the Security to Group table at compile-time by adding an entry in `Snm_Cfg_Sec2Groups` in the *networking/snmp/snmp\_cfg.c* file and modifying the value of `VACM_SEC2GRP_TBL_SIZE` in the *include/networking/snmp\_cfg.h* file. Refer to “[Advanced SNMP Topics](#)” on page 1693 for more information on the VACM Security to Group table.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Security to Group table.

#### Note



The *snmp\_api.h* file needs to be included.

```
UINT8 security_name[] = "user1";
UINT8 group_name[]   = "user_grp";
VACM_SEC2GROUP_STRUCT node;
VACM_SEC2GROUP_STRUCT *location_ptr;

/* Check the limits of the context_name. */
if ((strlen ((CHAR*) security_name) >= SNMP_SIZE_SMALLOBJECTID)
```

```
    || (strlen ((CHAR*) group_name) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the structure. */
    memset (&node, 0, sizeof(VACM_SEC2GROUP_STRUCT));

    /* Fill in the structure. */
    node.vacm_security_model = SNMP_CBSM_V1;
    strcpy ((CHAR *) node.vacm_security_name,
            (CHAR*) security_name);
    strcpy ((CHAR *) node.vacm_group_name, (CHAR*) group_name);

#ifdef INCLUDE_MIB_VACM == NU_TRUE
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status       = SNMP_ROW_ACTIVE;
#endif

    /* Call the function to insert the node. */
    VACM_InsertGroup (&node);
}

/* Find the group. */
VACM_Search_Group (SNMP_CBSM_V1, security_name, &location_ptr);

#ifdef SNMP_ENABLE_FILE_STORAGE == NU_TRUE
    VACM_Save_Group (location_ptr);
#endif

/* Remove the group. */
VACM_Remove_Group (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_Group](#)

[VACM\\_SEC2GROUP\\_STRUCT](#)



## VACM\_Search\_Group

This function returns a pointer to the Security to Group table entry corresponding to the passed parameters.

### Usage

```
STATUS VACM_Search_Group (UINT32          snmp_sm,  
                        UINT8          *security_name,  
                        VACM_SEC2GROUP_STRUCT **location_ptr);
```

### Arguments

- `snmp_sm`  
Security model identifier.
- `security_name`  
Pointer to the security name.
- `location_ptr`  
Double pointer to the Security to Group entry.  
Data structure [VACM\\_SEC2GROUP\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
The Security to Group table entry was found.
- `SNMP_ERROR`  
The Security to Group table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Security to Group table.

#### Note



The *snmp\_api.h* file needs to be included.

```
UINT8 security_name[] = "user1";  
UINT8 group_name[]   = "user_grp";  
VACM_SEC2GROUP_STRUCT node;  
VACM_SEC2GROUP_STRUCT *location_ptr;  
  
/* Check the limits of the context_name. */  
if ((strlen ((CHAR*) security_name) >= SNMP_SIZE_SMALLOBJECTID)  
    || (strlen ((CHAR*) group_name) >= SNMP_SIZE_SMALLOBJECTID))  
{  
    status = SNMP_ERROR;  
}
```

```
else
{
    /* Clear the structure. */
    memset (&node, 0, sizeof(VACM_SEC2GROUP_STRUCT));

    /* Fill in the structure. */
    node.vacm_security_model = SNMP_CBSM_V1;
    strcpy ((CHAR *) node.vacm_security_name,
            (CHAR*) security_name);
    strcpy ((CHAR *) node.vacm_group_name, (CHAR*) group_name);

    #if (INCLUDE_MIB_VACM == NU_TRUE)
        node.vacm_storage_type    = SNMP_STORAGE_NONVOLATILE;
        node.vacm_status          = SNMP_ROW_ACTIVE;
    #endif

    /* Call the function to insert the node. */
    VACM_InsertGroup (&node);
}

/* Find the group. */
VACM_Search_Group (SNMP_CBSM_V1, security_name, &location_ptr);

#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_Group (location_ptr);
#endif

/* Remove the group. */
VACM_Remove_Group (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_InsertGroup](#)

[VACM\\_SEC2GROUP\\_STRUCT](#)

## VACM\_Remove\_Group

This function removes an entry from the Security to Group table. The entry is also removed from the file system, if it was previously stored there, and the memory previously allocated for this entry is deallocated.

### Usage

```
STATUS VACM_Remove_Group (VACM_SEC2GROUP_STRUCT *location_ptr);
```

### Arguments

- location\_ptr

Pointer to the entry to be removed.

Data structure [VACM\\_SEC2GROUP\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

The Security to Group table entry was removed successfully.

- SNMP\_ERROR

Invalid argument, or the Security to Group table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Security to Group table.

#### Note



The *snmp\_api.h* file needs to be included.

```
UINT8 security_name[] = "user1";
UINT8 group_name[]    = "user_grp";
VACM_SEC2GROUP_STRUCT node;
VACM_SEC2GROUP_STRUCT *location_ptr;

/* Check the limits of the context_name. */
if ((strlen ((CHAR*) security_name) >= SNMP_SIZE_SMALLOBJECTID)
    || (strlen ((CHAR*) group_name) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the structure. */
    memset (&node, 0, sizeof(VACM_SEC2GROUP_STRUCT));

    /* Fill in the structure. */
    node.vacm_security_model = SNMP_CBSM_V1;
    strcpy ((CHAR *) node.vacm_security_name,
```

```
        (CHAR*) security_name);
    strcpy ((CHAR *) node.vacm_group_name, (CHAR*) group_name);

    #if (INCLUDE_MIB_VACM == NU_TRUE)
        node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
        node.vacm_status       = SNMP_ROW_ACTIVE;
    #endif

    /* Calling the function to insert the node. */
    VACM_InsertGroup (&node);
}

/* Find the group. */
VACM_Search_Group (SNMP_CBSM_V1, security_name, &location_ptr);

    #if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
        VACM_Save_Group (location_ptr);
    #endif

    /* Remove the group. */
    VACM_Remove_Group (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_InsertGroup](#)

[VACM\\_SEC2GROUP\\_STRUCT](#)

## VACM\_Save\_Group

This function saves an entry from the VACM Security to Group table to the file system.

---

**Note**

SNMP must be configured to enable storage of data to the file system.

---

### Usage

```
STATUS VACM_Save_Group (VACM_SEC2GROUP_STRUCT *location_ptr);
```

### Arguments

- location\_ptr

Pointer to the entry to be saved in the file system.

Data structure [VACM\\_SEC2GROUP\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

The Security to Group table entry was saved successfully.

### Description

This function needs to be invoked when a new node has been added and the node needs to be saved to the file. It will also need to be called after the values of an entry have been updated.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Security to Group table.

---

**Note**

The *snmp\_api.h* file needs to be included.

---

```
UINT8 security_name[] = "user1";
UINT8 group_name[]    = "user_grp";
VACM_SEC2GROUP_STRUCT node;
VACM_SEC2GROUP_STRUCT *location_ptr;

/* Check the limits of the context_name. */
if ((strlen ((CHAR*) security_name) >= SNMP_SIZE_SMALLOBJECTID)
    || (strlen ((CHAR*) group_name) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the structure. */
    memset (&node, 0, sizeof(VACM_SEC2GROUP_STRUCT));
```

```
/* Fill in the structure. */
node.vacm_security_model = SNMP_CBSM_V1;
strcpy ((CHAR *) node.vacm_security_name,
        (CHAR*) security_name);
strcpy ((CHAR *) node.vacm_group_name, (CHAR*) group_name);

#if (INCLUDE_MIB_VACM == NU_TRUE)
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status       = SNMP_ROW_ACTIVE;
#endif

/* Call the function to insert the node. */
VACM_InsertGroup (&node);
}

/* Find the group. */
VACM_Search_Group (SNMP_CBSM_V1, security_name, &location_ptr);

#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_Group (location_ptr);
#endif

/* Remove the group. */
VACM_Remove_Group (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_Group](#)

[VACM\\_SEC2GROUP\\_STRUCT](#)

## VACM\_InsertAccessEntry

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS VACM_InsertAccessEntry (VACM_ACCESS_STRUCT *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [VACM\\_ACCESS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added to the VACM Access table.
- SNMP\_ERROR  
Invalid argument, or the VACM Access table entry was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the VACM Access table. This function does not save the node to the file system. If the node needs to be added to the file system, [VACM\\_Save\\_Access](#) must be invoked. An entry can also be added to the VACM Access table at compile-time by adding an entry in `Snm_Cfg_Access` in the *networking/snmp/snmp\_cfg.c* file and modifying the value of `VACM_ACCESS_TBL_SIZE` in the *include/networking/snmp\_cfg.h* file. Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on the VACM Access table.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Access table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
VACM_ACCESS_STRUCT      node;
VACM_ACCESS_STRUCT      *location_ptr = NU_NULL;
/* Under the default configuration, the empty string context name is
 * registered with Nucleus SNMP.
 */
UINT8* context_prefix = "";
UINT8* group_name = "group100";
UINT8* read_view = "readview";
```

```
UINT8* write_view = "writeview";
UINT8* notify_view = "notifyview";

/* If any of the conditions are false then return an error. */
if( (strlen((CHAR*)group_name) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)context_prefix) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)read_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)write_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)notify_view) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the node. */
    memset (&node, 0, sizeof(VACM_ACCESS_STRUCT));

    /* Fill the node. */
    strcpy ((CHAR *) node.vacm_group_name, (CHAR *) group_name);
    strcpy ((CHAR *) node.vacm_context_prefix,
            (CHAR *) context_prefix);
    node.vacm_security_model = SNMP_CBSM_V1;
    node.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;
    node.vacm_context_match = VACM_CTXTMATCH_EXACT;
    strcpy ((CHAR *) node.vacm_read_view, (CHAR *) read_view);
    strcpy ((CHAR *) node.vacm_write_view, (CHAR *) write_view);
    strcpy ((CHAR *) node.vacm_notify_view, (CHAR *) notify_view);
#ifdef INCLUDE_MIB_VACM == NU_TRUE
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status = SNMP_ROW_ACTIVE;
#endif

    /* Call the function to insert the node. */
    VACM_InsertAccessEntry (&node);
}

/* Find the Access Entry. */
VACM_Search_AccessEntry (group_name, context_prefix, SNMP_CBSM_V1,
                        SNMP_SECURITY_NOAUTHNOPRIV, &location_ptr);

#ifdef SNMP_ENABLE_FILE_STORAGE == NU_TRUE
    VACM_Save_Access (location_ptr);
#endif
/* Remove the Access Entry. */
VACM_Remove_AccessEntry (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_AccessEntry](#)

[VACM\\_ACCESS\\_STRUCT](#)



## VACM\_Search\_AccessEntry

This function returns a pointer to an entry for the VACM Access table corresponding to the passed parameters.

### Usage

```
STATUS VACM_Search_AccessEntry (UINT8          *group_name,  
                                UINT8          *context_prefix,  
                                UINT32         snmp_sm,  
                                UINT8         security_level,  
                                VACM_ACCESS_STRUCT **location_ptr);
```

### Arguments

- **group\_name**  
Pointer to the group name.
- **context\_prefix**  
Pointer to the context prefix.
- **snmp\_sm**  
The security model identifier. It can have following values:
  - SNMP\_CBSM\_V1 : For CBSMv1
  - SNMP\_CBSM\_V2 : For CBSMv2
  - SNMP\_USM : For USM
- **security\_level**  
The security level. It can have following values:
  - SNMP\_SECURITY\_NOAUTHNOPRIV : No authentication and no privacy.
  - SNMP\_SECURITY\_AUTHNOPRIV : Authentication and no privacy.
  - SNMP\_SECURITY\_AUTHPRIV : Authentication with privacy.
- **location\_ptr**  
A double pointer to the VACM Access table entry.  
Data structure [VACM\\_ACCESS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
The VACM Access table entry was found.
- **SNMP\_ERROR**  
The VACM Access table entry was not found.

## Example

The following is an example for adding, removing, finding and saving entries to the VACM Access table.

### Note



The *snmp\_api.h* file needs to be included.

```
VACM_ACCESS_STRUCT      node;
VACM_ACCESS_STRUCT      *location_ptr = NU_NULL;
/* Under the default configuration the empty string context name is
 * registered with Nucleus SNMP.
 */
UINT8* context_prefix = "";
UINT8* group_name = "group100";
UINT8* read_view = "readview";
UINT8* write_view = "writeview";
UINT8* notify_view = "notifyview";

/* If any of the conditions is false, an return error. */
if( (strlen((CHAR*)group_name) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)context_prefix) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)read_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)write_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)notify_view) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the node. */
    memset (&node, 0, sizeof(VACM_ACCESS_STRUCT));

    /* Fill the node. */
    strcpy ((CHAR *) node.vacm_group_name, (CHAR *) group_name);
    strcpy ((CHAR *) node.vacm_context_prefix,
            (CHAR *) context_prefix);
    node.vacm_security_model = SNMP_CBSM_V1;
    node.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;
    node.vacm_context_match = VACM_CTXTMATCH_EXACT;
    strcpy ((CHAR *) node.vacm_read_view, (CHAR *) read_view);
    strcpy ((CHAR *) node.vacm_write_view, (CHAR *) write_view);
    strcpy ((CHAR *) node.vacm_notify_view, (CHAR *) notify_view);
#if (INCLUDE_MIB_VACM == NU_TRUE)
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status = SNMP_ROW_ACTIVE;
#endif

    /* Calling the function to insert the node. */
    VACM_InsertAccessEntry (&node);
}

/* Find the Access Entry. */
VACM_Search_AccessEntry (group_name, context_prefix, SNMP_CBSM_V1,
                        SNMP_SECURITY_NOAUTHNOPRIV, &location_ptr);
```

```
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_Access (location_ptr);
#endif
/* Remove the Access Entry. */
VACM_Remove_AccessEntry (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[VACM\\_InsertAccessEntry](#)

[SNMP Data Structures](#)

[VACM\\_ACCESS\\_STRUCT](#)

## VACM\_Remove\_AccessEntry

This function removes an entry from the VACM Access table. The entry is also removed from the file system, if it was previously stored there, and the memory previously allocated for this entry is deallocated.

### Usage

```
STATUS VACM_Remove_AccessEntry (VACM_ACCESS_STRUCT *location_ptr);
```

### Arguments

- `location_ptr`

Pointer to the entry to be removed.

Data structure [VACM\\_ACCESS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`

The VACM Access table entry was removed successfully.

- `SNMP_ERROR`

Invalid argument, or the VACM Access table entry was not found.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Access table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
VACM_ACCESS_STRUCT      node;
VACM_ACCESS_STRUCT      *location_ptr = NU_NULL;
/* Under the default configuration, the empty string context name is
 * registered with Nucleus SNMP.
 */
UINT8* context_prefix = "";
UINT8* group_name = "group100";
UINT8* read_view = "readview";
UINT8* write_view = "writeview";
UINT8* notify_view = "notifyview";

/* If any of the conditions are false, return an error. */
if( (strlen((CHAR*)group_name) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)context_prefix) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)read_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)write_view) >= SNMP_SIZE_SMALLOBJECTID) ||
    (strlen((CHAR*)notify_view) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
```

```
}
else
{
    /* Clear the node. */
    memset (&node, 0, sizeof(VACM_ACCESS_STRUCT));

    /* Fill the node. */
    strcpy ((CHAR *) node.vacm_group_name, (CHAR *) group_name);
    strcpy ((CHAR *) node.vacm_context_prefix,
            (CHAR *) context_prefix);
    node.vacm_security_model = SNMP_CBSM_V1;
    node.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;
    node.vacm_context_match = VACM_CTXTMATCH_EXACT;
    strcpy ((CHAR *) node.vacm_read_view, (CHAR *) read_view);
    strcpy ((CHAR *) node.vacm_write_view, (CHAR *) write_view);
    strcpy ((CHAR *) node.vacm_notify_view, (CHAR *) notify_view);
    #if (INCLUDE_MIB_VACM == NU_TRUE)
        node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
        node.vacm_status = SNMP_ROW_ACTIVE;
    #endif

    /* Call the function to insert the node. */
    VACM_InsertAccessEntry (&node);
}

/* Find the Access Entry. */
VACM_SearchAccessEntry (group_name, context_prefix, SNMP_CBSM_V1,
                        SNMP_SECURITY_NOAUTHNOPRIV, &location_ptr);

#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_SaveAccess (location_ptr);
#endif
/* Remove the Access Entry. */
VACM_RemoveAccessEntry (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_InsertAccessEntry](#)

[VACM\\_ACCESS\\_STRUCT](#)

## VACM\_Save\_Access

This function saves an entry from the VACM Access table to the file system.

### Note



SNMP must be configured to enable storage of data to the file system.

### Usage

```
STATUS VACM_Save_Access (VACM_ACCESS_STRUCT *vacm_access);
```

### Arguments

- vacm\_access

Pointer to the entry to be saved.

Data structure [VACM\\_ACCESS\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

The VACM Access table entry was saved successfully.

### Description

This function needs to be invoked when a new node has been added, and the node needs to be saved to a file. It also needs to be called after the values of an entry have been updated.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM Access table.

### Note



The *snmp\_api.h* file needs to be included.

```
VACM_ACCESS_STRUCT      node;
VACM_ACCESS_STRUCT      *location_ptr = NU_NULL;
/* Under the default configuration, the empty string context name is
 * registered with Nucleus SNMP.
 */
UINT8* context_prefix = "";
UINT8* group_name = "group100";
UINT8* read_view = "readview";
UINT8* write_view = "writeview";
UINT8* notify_view = "notifyview";

/* If any of the conditions are false, return an error. */
```

```
if( (strlen((CHAR*)group_name) >= SNMP_SIZE_SMALLOBJECTID)      ||
    (strlen((CHAR*)context_prefix) >= SNMP_SIZE_SMALLOBJECTID)  ||
    (strlen((CHAR*)read_view) >= SNMP_SIZE_SMALLOBJECTID)       ||
    (strlen((CHAR*)write_view) >= SNMP_SIZE_SMALLOBJECTID)      ||
    (strlen((CHAR*)notify_view) >= SNMP_SIZE_SMALLOBJECTID))
{
    status = SNMP_ERROR;
}
else
{
    /* Clear the node. */
    memset (&node, 0, sizeof(VACM_ACCESS_STRUCT));

    /* Fill the node. */
    strcpy ((CHAR *) node.vacm_group_name, (CHAR *) group_name);
    strcpy ((CHAR *) node.vacm_context_prefix,
            (CHAR *) context_prefix);
    node.vacm_security_model = SNMP_CBSM_V1;
    node.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;
    node.vacm_context_match = VACM_CTXTMATCH_EXACT;
    strcpy ((CHAR *) node.vacm_read_view, (CHAR *) read_view);
    strcpy ((CHAR *) node.vacm_write_view, (CHAR *) write_view);
    strcpy ((CHAR *) node.vacm_notify_view, (CHAR *) notify_view);
#ifdef INCLUDE_MIB_VACM == NU_TRUE
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status = SNMP_ROW_ACTIVE;
#endif

    /* Call the function to insert the node. */
    VACM_InsertAccessEntry (&node);
}

/* Find the Access Entry. */
VACM_SearchAccessEntry (group_name, context_prefix, SNMP_CBSM_V1,
                        SNMP_SECURITY_NOAUTHNOPRIV, &location_ptr);

#ifdef SNMP_ENABLE_FILE_STORAGE == NU_TRUE
    VACM_Save_Access (location_ptr);
#endif
/* Remove the Access Entry. */
VACM_RemoveAccessEntry (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[VACM\\_Search\\_AccessEntry](#)

[SNMP Data Structures](#)

[VACM\\_ACCESS\\_STRUCT](#)

## VACM\_InsertMibView

This function allocates memory and then copies the values from the passed node.

### Usage

```
STATUS VACM_InsertMibView (VACM_VIEWTREE_STRUCT *node);
```

### Arguments

- node  
Pointer to the entry to be added.  
Data structure [VACM\\_VIEWTREE\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The entry was added to the VACM MIB View table.
- SNMP\_ERROR  
Invalid argument, or the VACM MIB View table entry was already registered.
- NU\_NO\_MEMORY  
Memory allocation failed.

### Description

The newly allocated node is added to the VACM MIB View table. This function does not save the node to the file system. If the node needs to be added to the file system, [VACM\\_Save\\_View](#) must be invoked. An entry can also be added to the VACM MIB View table at compile-time by adding an entry in `Snmp_Cfg_Mib_View` in the *networking/snmp/snmp\_cfg.c* file and modifying the value of `VACM_MIB_VIEW_TBL_SIZE` in the *include/networking/snmp\_cfg.h* file. Refer to “[Advanced SNMP Topics](#)” on page 1693 for more information on VACM MIB View table.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM View Tree Family table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
/* VACM MIB View that would only have access to enterprises MIB.
 */
VACM_VIEWTREE_STRUCT vacm_mib_view;

/* Accessible Subtree view. The value of 1.3.6.1.4.1 would also
 * include 1.3.6.1.4.1.x.x.x.x. ...
 */
```



```
UINT32 accessible_subtree[] = {1,3,6,1,4,1};

/* Length of Subtree */
UINT32 accessible_subtree_len = 6;

/* Family mask must be 6 bits long. For wildcards, the bit is set to 0,
 * all other bits to be checked must be set to 1.
 * Consider a case, when you need to access all Managed
 * Objects in a subtree with an OID such as 1.3.6.1.x.1. For this
 * case you need the family mask with a value of 111101 binary hex
 * equivalent is {0xF4}.
 */
UINT8 subtree_family_mask[] = {0xFC};

/* The mask length in bytes would be 1, since 6 bits can't be
 * represented in fewer bytes.
 */
UINT32 subtree_family_mask_len = 1;

/* View name */
CHAR view_name[] = "OnlyEnterprisesAccess";

/* Handle to the VACM MIB View */
VACM_VIEWTREE_STRUCT *location_ptr = NU_NULL;

/* Status to return success or error code */
STATUS status;

/* Clear out the structure. */
memset (&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the value of View Name. */
strcpy ((CHAR *) vacm_mib_view.vacm_view_name, view_name);

/* Set the value to Subtree. */
memcpy (vacm_mib_view.vacm_subtree, accessible_subtree,
        sizeof(accessible_subtree));

/* Set the Subtree length. */
vacm_mib_view.vacm_subtree_len = accessible_subtree_len;

/* Set the value for the family mask. */
memcpy (vacm_mib_view.vacm_family_mask, subtree_family_mask,
        sizeof(subtree_family_mask));

/* Set the value to family mask length. */
vacm_mib_view.vacm_mask_length = subtree_family_mask_len;

/* Allow access to the subtree. */
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set storage type. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the row status to 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
#endif
```

```
/* Add a VACM MIB View Entry. */
status = VACM_InsertMibView(&vacm_mib_view);

/* Find View. */
VACM_Search_MibView (view_name, (UINT32*) accessible_subtree,
                    accessible_subtree_len, &location_ptr);

/* Save View. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_View (location_ptr);
#endif

/* Remove View. */
VACM_Remove_MibView (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Search\\_MibView](#)

[VACM\\_VIEWTREE\\_STRUCT](#)

## VACM\_Search\_MibView

This function returns a pointer to an entry for the VACM MIB View table corresponding to the passed parameters.

### Usage

```
STATUS VACM_Search_MibView (UINT8          *view_name,  
                           UINT32         *subtree,  
                           UINT32         subtree_len,  
                           VACM_VIEWTREE_STRUCT **location_ptr);
```

### Arguments

- `view_name`  
Pointer to view name.
- `subtree`  
Pointer to sub-tree.
- `subtree_len`  
Length of sub-tree.
- `location_ptr`  
Double pointer to the VACM MIB View entry.  
  
Data structure [VACM\\_VIEWTREE\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
The VACM MIB View table entry was found.
- `SNMP_ERROR`  
Invalid arguments, or the VACM MIB View table entry does not exist.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM View Tree Family table.

---

**Note**

The *snmp\_api.h* file needs to be included.

---

```
/* VACM MIB View that would only have access to enterprises MIB.  
 */  
VACM_VIEWTREE_STRUCT vacm_mib_view;  
  
/* Accessible Subtree view. The value of 1.3.6.1.4.1 would also  
 * include 1.3.6.1.4.1.x.x.x.x. ...  
 */
```

```
UINT32 accessible_subtree[] = {1,3,6,1,4,1};

/* Length of Subtree */
UINT32 accessible_subtree_len = 6;

/* The family mask must be 6 bits long. For wildcards, the bit is
 * set to 0, all other bits to be checked must be set to 1.
 * Consider a case, when you need to access all Managed
 * Objects in a subtree with an OID such as 1.3.6.1.x.1. For this
 * case you need the family mask with a value of 111101 binary hex
 * equivalent is {0xF4}.
 */
UINT8 subtree_family_mask[] = {0xFC};

/* Mask length in bytes would be 1, since 6 bits can't be
 * represented in fewer bytes.
 */
UINT32 subtree_family_mask_len = 1;

/* View name */
CHAR view_name[] = "OnlyEnterprisesAccess";

/* Handle to the VACM MIB View */
VACM_VIEWTREE_STRUCT *location_ptr = NU_NULL;

/* Status to return success or error code */
STATUS status;

/* Clear out the structure. */
memset (&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the value of View Name. */
strcpy ((CHAR *) vacm_mib_view.vacm_view_name, view_name);

/* Set the value to Subtree. */
memcpy (vacm_mib_view.vacm_subtree, accessible_subtree,
        sizeof(accessible_subtree));

/* Set the Subtree length. */
vacm_mib_view.vacm_subtree_len = accessible_subtree_len;

/* Set the value of family mask. */
memcpy (vacm_mib_view.vacm_family_mask, subtree_family_mask,
        sizeof(subtree_family_mask));

/* Set the value to family mask length. */
vacm_mib_view.vacm_mask_length = subtree_family_mask_len;

/* Allow access to the subtree. */
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the storage type. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the row status to 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
```

```
/* Add an MIB View Entry. */
status = VACM_InsertMibView(&vacm_mib_view);

/* Find View. */
VACM_Search_MibView (view_name, (UINT32*) accessible_subtree,
                    accessible_subtree_len, &location_ptr);

/* Save View. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_View (location_ptr);
#endif

/* Remove View. */
VACM_Remove_MibView (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_MibView](#)

[VACM\\_VIEWTREE\\_STRUCT](#)

## VACM\_Remove\_MibView

This function removes an entry from the VACM MIB View table. The entry is also removed from the file system, if it was previously stored there, and the memory previously allocated for this entry is deallocated.

### Usage

```
STATUS VACM_Remove_MibView (VACM_VIEWTREE_STRUCT *location_ptr);
```

### Arguments

- `location_ptr`

Pointer to the entry to be removed.

Data structure [VACM\\_VIEWTREE\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`

The VACM MIB View table entry was removed successfully.

- `SNMP_ERROR`

Invalid arguments, or the VACM MIB View table entry does not exist.

### Example

The following is an example for adding, removing, finding and saving entries to the VACM View Tree Family table.

---

#### Note



The *snmp\_api.h* file needs to be included.

---

```
/* VACM MIB View that would only have access to enterprises MIB.
 */
VACM_VIEWTREE_STRUCT vacm_mib_view;

/* Accessible Subtree view. The value of 1.3.6.1.4.1 would also
 * include 1.3.6.1.4.1.x.x.x.x. ...
 */
UINT32 accessible_subtree[] = {1,3,6,1,4,1};

/* Length of Subtree */
UINT32 accessible_subtree_len = 6;

/* The family mask must be 6 bits long. For wildcards, the bit is
 * set to 0, all other bits to be checked must be set to 1.
 * Consider a case, when you need to access all Managed
 * Objects in a subtree with an OID such as 1.3.6.1.x.1. For this
 * case you need the family mask with a value of 111101 binary hex
 * equivalent is {0xF4}.
 */
```

```
UINT8 subtree_family_mask[] = {0xFC};

/* Mask length in bytes would be 1, since 6 bits can't be
 * represented in fewer bytes.
 */
UINT32 subtree_family_mask_len = 1;

/* View name */
CHAR view_name[] = "OnlyEnterprisesAccess";

/* Handle to the VACM MIB View */
VACM_VIEWTREE_STRUCT *location_ptr = NU_NULL;

/* Status to return success or error code */
STATUS status;

/* Clear out the structure. */
memset (&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the value of View Name. */
strcpy ((CHAR *) vacm_mib_view.vacm_view_name, view_name);

/* Set the value to Subtree. */
memcpy (vacm_mib_view.vacm_subtree, accessible_subtree,
        sizeof(accessible_subtree));

/* Set the Subtree length. */
vacm_mib_view.vacm_subtree_len = accessible_subtree_len;

/* Set the value of family mask. */
memcpy (vacm_mib_view.vacm_family_mask, subtree_family_mask,
        sizeof(subtree_family_mask));

/* Set the value to family mask length. */
vacm_mib_view.vacm_mask_length = subtree_family_mask_len;

/* Allow access to the subtree. */
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the storage type. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the row status to 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add a VACM MIB View Entry. */
status = VACM_InsertMibView(&vacm_mib_view);

/* Find View. */
VACM_Search_MibView (view_name, (UINT32*) accessible_subtree,
                    accessible_subtree_len, &location_ptr);

/* Save View. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
```

```
VACM_Save_View (location_ptr);  
#endif  
  
/* Remove View. */  
VACM_Remove_MibView (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_InsertMibView](#)

[VACM\\_VIEWTREE\\_STRUCT](#)



## VACM\_Save\_View

This function saves an entry from the VACM View Tree table to the file system.

---

**Note**

---



SNMP must be configured to enable storage of data to the file system.

---

### Usage

```
STATUS VACM_Save_View (VACM_VIEWTREE_STRUCT *vacm_view_tree_family);
```

### Arguments

- vacm\_view\_tree\_family

Pointer to the entry to be saved.

Data structure [VACM\\_VIEWTREE\\_STRUCT](#) is defined in the [SNMP Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

The VACM MIB View table entry was saved successfully.

### Description

This function needs to be invoked when a new node has been added and the node needs to be saved to a file. It also needs to be called after the values of an entry have been updated.

### Example

The following is an example for adding, removing, finding and saving entries to VACM View Tree Family table.

---

**Note**

---



The *snmp\_api.h* file needs to be included.

---

```
UINT32 subtree[] = {1, 3, 6, 1, 2, 1, 1};
INT8 subtree_len = 7;
INT8 mask_len = 1;
UINT8 family_mask[1] = {0xFE};
UINT8 *view_name = "zeus";
VACM_VIEWTREE_STRUCT node;
VACM_VIEWTREE_STRUCT *location_ptr = NU_NULL;

/* If any of the conditions are true, return an error. */
if ((strlen ((CHAR*) view_name) >= SNMP_SIZE_SMALLOBJECTID) ||
    (mask_len >= VACM_MASK_SIZE))
{
    status = SNMP_ERROR;
}
else
{

```

```
/* Clear the structure. */
memset (&node, 0, sizeof(VACM_VIEWTREE_STRUCT));

/* Fill in the structure. */
strcpy ((CHAR *) node.vacm_view_name, (CHAR *) view_name);
memcpy (node.vacm_subtree, subtree,
        (unsigned int)(sizeof(UINT32) * subtree_len));
node.vacm_subtree_len = (UINT16) subtree_len;
node.vacm_family_type = VACM_FAMILY_INCLUDED;

if (family_mask != NU_NULL)
    memcpy (node.vacm_family_mask,
            family_mask, (unsigned int) mask_len);

node.vacm_mask_length = mask_len;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    node.vacm_storage_type = SNMP_STORAGE_NONVOLATILE;
    node.vacm_status       = SNMP_ROW_ACTIVE;
#endif

/* Call the function to insert node. */
VACM_InsertMibView (&node);
}

/* Find View. */
VACM_Search_MibView (view_name, (UINT32*) subtree, subtree_len,
                    &location_ptr);

/* Save View. */
#if (SNMP_ENABLE_FILE_STORAGE == NU_TRUE)
    VACM_Save_View (location_ptr);
#endif

/* Remove View. */
VACM_Remove_MibView (location_ptr);
```

## Related Topics

[SNMP API Functions](#)

[SNMP Data Structures](#)

[VACM\\_Remove\\_MibView](#)

[VACM\\_VIEWTREE\\_STRUCT](#)

## Nucleus MIB-II Defines

The Management Information Base for Network Management of TCP/IP-based internets (MIB-II), as defined in RFC 1213 and RFC 2863, is built into the SNMP library by default. You have the option to remove particular components from the build by setting defines in the [RFC1213 Defines File](#).

### RFC1213 Defines File

The user-definable region for MIB-II is provided in the *rfc1213/inc/1213xxxx.h* file. Although you may define other variables, normally only those within this definable region are required.

The following defines enable you to exclude MIB-II components from Nucleus SNMP. By default, the EGP component is excluded from the build because Nucleus NET does not support EGP.

### RFC1213\_IP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the IP component from the build. If using Nucleus NET version 5.2, `MIB2_IP_INCLUDE` in */include/networking/net\_cfg.h* may also be set to `NU_FALSE` when excluding the IP component from the build to reduce the build size and improve efficiency.

### RFC1213\_IF\_INCLUDE

Setting this macro to `NU_FALSE` excludes the Interfaces component from the build. If using Nucleus NET version 5.2, `MIB2_IF_INCLUDE` in */include/networking/net\_cfg.h* may also be set to `NU_FALSE` when excluding the interface component from the build to reduce the build size and improve efficiency.

### RFC1213\_TCP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the TCP component from the build. If using Nucleus NET version 5.2, `MIB2_TCP_INCLUDE` in */include/networking/net\_cfg.h* may also be set to `NU_FALSE` when excluding the TCP component from the build to reduce the build size and improve efficiency.

### RFC1213\_UDP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the UDP component from the build. If using Nucleus NET version 5.2, `MIB2_UDP_INCLUDE` in *net/net\_cfg.h* may also be set to `NU_FALSE` when excluding the UDP component from the build to reduce the build size and improve efficiency.

## RFC1213\_SYS\_INCLUDE

Setting this macro to `NU_FALSE` excludes the System component from the build. If using Nucleus NET version 5.2, `MIB2_SYS_INCLUDE` in `/include/networking/net_cfg.h` may also be set to `NU_FALSE` when excluding the Interface component from the build to reduce the build size and improve efficiency.

## RFC1213\_EGP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the EGP component from the build. This macro is set to `NU_FALSE` by default since Nucleus NET does not support the EGP.

## RFC1213\_ICMP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the ICMP component from the build. If using Nucleus NET version 5.2, `MIB2_ICMP_INCLUDE` in `/include/networking/net_cfg.h` may also be set to `NU_FALSE` when excluding the ICMP component from the build to reduce the build size and improve efficiency.

## RFC1213\_SNMP\_INCLUDE

Setting this macro to `NU_FALSE` excludes the MIB-2 SNMP component from the build.

## INCLUDE\_SNMPv1\_FULL\_MIBII

Setting this macro to `NU_FALSE` excludes the IP, TCP, UDP, EGP, ICMP and MIB-2 SNMP components from the build.

# MIB-II: Managed Objects

## The System Group

The members of the System Group are initialized to the values stored in the MIB-II macros in the `include/networking/snmp_prt.h` file.

**Table 9-24. System Group**

Member	Access	Description
sysDescr	READ-ONLY	The full name and version identification of the system's hardware type, software operating system, and networking software.
sysObjectID	READ-ONLY	The vendor's authoritative identification of the network management subsystem contained in the entity.

**Table 9-24. System Group (cont.)**

Member	Access	Description
sysUpTime	READ-ONLY	The time (in hundredths of a second) since the network management portion of the system was last re-initialized.
sysContact	READ-ONLY	The contact person for this managed node, including his or her contact information.
sysName	READ-WRITE	An administratively-assigned name for this managed node. By convention, this is the node's fully-qualified domain name.
sysLocation	READ-WRITE	The physical location of this node.
sysServices	READ-ONLY	A sum that indicates the set of services this entity primarily offers.

## The Interfaces Group

The Interfaces Group contains statistics about the network interfaces on which this system can send and receive IP datagrams.

**Table 9-25. Interfaces Group**

Member	Access	Description
ifNumber	READ-ONLY	The number of network interfaces present on this system.
ifTable	NOT-ACCESSIBLE	A list of interface entries.
ifEntry	NOT-ACCESSIBLE	An interface entry.
ifIndex	READ-ONLY	A unique value for each interface. Its value ranges between 1 and the value of ifNumber.
ifDescr	READ-ONLY	The name of the manufacturer, the product name and the version of the hardware interface.
ifType	READ-ONLY	The type of interface.
ifMtu	READ-ONLY	The size of the largest datagram which can be sent/received on the interface, specified in octets.
ifSpeed	READ-ONLY	An estimate of the interface's current bandwidth in bits per second.

**Table 9-25. Interfaces Group (cont.)**

Member	Access	Description
ifPhysAddress	READ-ONLY	The interface's address at the protocol layer immediately 'below' the network layer in the protocol stack.
ifAdminStatus	READ-WRITE	The desired state of the interface.
ifOperStatus	READ-ONLY	The current operational state of the interface.
ifLastChange	READ-ONLY	The value of sysUpTime at the time the interface entered its current operational state.
ifInOctets	READ-ONLY	The total number of octets received on the interface, including framing characters.
ifInUcastPkts	READ-ONLY	The number of subnetwork-unicast packets delivered to a higher-layer protocol.
ifInNUcastPkts	READ-ONLY	The number of non-unicast packets delivered to a higher-layer protocol.
ifInDiscards	READ-ONLY	The number of inbound packets that were chosen to be discarded. Packets are counted even if no errors had been detected to prevent their being deliverable to a higher-layer protocol.
ifInErrors	READ-ONLY	The number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.
ifInUnknownProtos	READ-ONLY	The number of packets received via interfaces that were discarded because of an unknown or unsupported protocol.
ifOutOctets	READ-ONLY	The total number of octets transmitted out of the interface, including framing characters.
ifOutUcastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted to a subnetwork-unicast address, including those that were discarded or not sent.
ifOutNUcastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted to a non-unicast address, including those that were discarded or not sent.
ifOutDiscards	READ-ONLY	The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent them from being transmitted.

**Table 9-25. Interfaces Group (cont.)**

Member	Access	Description
ifOutErrors	READ-ONLY	The number of outbound packets that could not be transmitted because of errors.
ifOutQLen	READ-ONLY	The length of the output packet queue (in packets).
ifSpecific	READ-ONLY	A reference to MIB definitions specific to the particular media being used to realize the interface. For example, if the interface is realized by an ethernet, then the value of this object refers to a document defining objects specific to ethernet. If this information is not present, its value should be set to the OBJECT IDENTIFIER {0 0}, which is a syntactically valid object identifier.

## The Interface Extension Group

The Interface Extension Group contains additional objects for the Interface Table.

**Table 9-26. Interface Extension Group**

Member	Access	Description
ifName	READ-ONLY	The textual name of the interface.
ifInMulticastPkts	READ-ONLY	The number of packets, delivered by this sub-layer to a higher (sub-) layer, which were addressed to a multicast address at this sub-layer. For a MAC layer protocol, this includes both Group and Functional addresses. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifInBroadcastPkts	READ-ONLY	The number of packets, delivered by this sub-layer to a higher (sub-) layer, which were addressed to a broadcast address at this sub-layer. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.

**Table 9-26. Interface Extension Group (cont.)**

Member	Access	Description
ifOutMulticastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted, and which were addressed to a multicast address at this sub-layer, including those that were discarded or not sent. For a MAC layer protocol, this includes both Group and Functional addresses. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifOutBroadcastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted, and which were addressed to a broadcast address at this sub-layer, including those that were discarded or not sent. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCInOctets	READ-ONLY	The total number of octets received on the interface, including framing characters. This object is a 64-bit version of ifInOctets. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCInUcastPkts	READ-ONLY	The number of packets, delivered by this sub-layer to a higher (sub-) layer, which were not addressed to a multicast or broadcast address at this sub-layer. This object is a 64-bit version of ifInUcastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter Discontinuity Time.



**Table 9-26. Interface Extension Group (cont.)**

Member	Access	Description
ifHCInMulticastPkts	READ-ONLY	The number of packets, delivered by this sub-layer to a higher (sub-) layer, which were addressed to a multicast address at this sub-layer. For a MAC layer protocol, this includes both Group and Functional addresses. This object is a 64-bit version of ifInMulticastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCInBroadcastPkts	READ-ONLY	The number of packets, delivered by this sub-layer to a higher (sub-) layer, which were addressed to a broadcast address at this sub-layer. This object is a 64-bit version of ifInBroadcastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCOctets	READ-ONLY	The total number of octets transmitted out of the interface, including framing characters. This object is a 64-bit version of ifOutOctets. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCOctetsUcastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted and were not addressed to a multicast or broadcast address at this sub-layer, including those that were discarded or not sent. This object is a 64-bit version of ifOut UcastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.

**Table 9-26. Interface Extension Group (cont.)**

Member	Access	Description
ifHCOutMulticastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted and were addressed to a multicast address at this sub-layer, including those that were discarded or not sent. For a MAC layer protocol, this includes both Group and Functional addresses. This object is a 64-bit version of ifOut MulticastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifHCOutBroadcastPkts	READ-ONLY	The total number of packets that higher-level protocols requested be transmitted and were addressed to a broadcast address at this sub-layer, including those that were discarded or not sent. This object is a 64-bit version of ifOut BroadcastPkts. Discontinuities in the value of this counter can occur at re-initialization of the management system, and at other times as indicated by the value of ifCounter DiscontinuityTime.
ifLinkUpDownTrapEnable	READ-WRITE	Indicates whether linkUp/linkDown traps should be generated for this interface. By default, this object should be enabled (set to '1') for interfaces which do not operate on 'top' of any other interface (as defined in the ifStackTable) and disabled (set to '2') otherwise.
ifHighSpeed	READ-ONLY	An estimate of the interface's current bandwidth in units of 1,000,000 bits per second. If this object reports a value of 'n' then the speed of the interface is somewhere in the range of 'n-500,000' to 'n+499,999'. For interfaces which do not vary in bandwidth, or for those where no accurate estimation can be made, this object should contain the nominal bandwidth. For a sub-layer which has no concept of bandwidth, this object should be zero.

**Table 9-26. Interface Extension Group (cont.)**

Member	Access	Description
ifPromiscuousMode	READ-ONLY	This object has a value of false ('2') if this interface only accepts packets/frames that are addressed to this station. It has a value of true ('1') when the station accepts all packets/frames transmitted on the media; however, the value of true is only legal on certain types of media. If legal, setting this object to true may require the interface to be reset before becoming effective. The value of this object does not affect the reception of broadcast and multicast packets/frames by the interface.
ifConnectorPresent	READ-ONLY	This object has the value of true ('1') if the interface sublayer has a physical connector and the value false ('2') otherwise.
ifAlias	READ-WRITE	<p>This object is an 'alias' name for the interface as specified by a network manager, and provides a non-volatile 'handle' for the interface.</p> <p>On the first instantiation of an interface, the value of ifAlias associated with that interface is the zero-length String. When a value is written into an instance of ifAlias through a network management set operation, the agent must retain the supplied value in the ifAlias instance associated with the same interface for as long as that interface remains instantiated. The ifAlias must be retained across all re-initializations/reboots of the network management system, including those which result in a change of the interface's ifIndex value. An example of the value which a network manager might store in this object for a WAN interface is the (Telco's) circuit number/identifier of the interface. Some agents may support write-access only for interfaces having particular values of ifType. An agent which supports write access to this object is required to keep the value in non-volatile storage; however, it may limit the length of new values depending on how much storage is already occupied by the current values for other interfaces.</p>

**Table 9-26. Interface Extension Group (cont.)**

Member	Access	Description
ifCounterDiscontinuityTime	READ-ONLY	The value of sysUpTime on the most recent occasion at which any one or more of this interface's counters suffered a discontinuity. The relevant counters are the specific instances associated with this interface of any Counter32 or Counter 64 object contained in the ifTable or ifXTable. If no such discontinuities have occurred since the last re-initialization of the local management subsystem, then this object contains a zero value.

## The Interface Stack Group

The Interface Stack Group contains information on the relationships between the multiple sub-layers of network interfaces.

**Table 9-27. Interface Stack Group**

Member	Access	Description
ifStackHigherLayer	NOT-ACCESSIBLE	The value of ifIndex corresponding to the higher sub-layer of the relationship, for example, the sub-layer which runs on 'top' of the sub-layer identified by the corresponding instance of ifStackLower Layer. If there is no higher sub-layer (below the internetwork layer), then this object has the value 0.
ifStackLowerLayer	NOT-ACCESSIBLE	The value of ifIndex corresponding to the lower sub-layer of the relationship, for example, the sub-layer which runs 'below' the sub-layer identified by the corresponding instance of ifStack HigherLayer. If there is no lower sub-layer, then this object has the value 0.
ifStackStatus	READ-CREATE	The status of the relationship between two sub-layers. Changing the value of this object from 'active' to 'notIn Service' or 'destroy' will likely have consequences up and down the interface stack.

## The Interface Receive Address Group

The Interface Receive Group contains an entry for each address (broadcast, multicast, or unicast) for which the system receives packets/frames on a particular interface.

**Table 9-28. Interface Receive Address Group**

Member	Access	Description
ifRcvAddressAddress	NOT-ACCESSIBLE	An address for which the system accepts packets/frames on this entry's interface.
ifRcvAddressStatus	READ-ONLY	This object is used to create and delete rows in the ifRcvAddress Table.

**Table 9-28. Interface Receive Address Group (cont.)**

Member	Access	Description
ifRcvAddressType	READ-ONLY	This object has the value nonVolatile ('3') for those entries in the table which are valid and are not be deleted by the next restart of the managed system. Entries having the value volatile ('2') are valid and exist, but have not been saved; therefore they will not exist after the next restart of the managed system. Entries having the value of other ('1') are valid and exist, but they are not classified as to whether they will continue to exist after the next restart.

## The IP Group

The IP Group contains various statistics and three tables: the [IP Group](#), the [ipRouteTable](#) and the [ipNetToMediaTable](#).

**Table 9-29. IP Group**

Member	Access	Description
ipForwarding	READ-WRITE	The indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity.
ipDefaultTTL	READ-WRITE	The default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity.
ipInReceives	READ-ONLY	The total number of input datagrams received from interfaces.
ipInHdrErrors	READ-ONLY	The number of input datagrams discarded due to errors in their IP headers.
ipInAddrErrors	READ-ONLY	The number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity.
ipForwDatagrams	READ-ONLY	The number of input datagrams for which this entity was not their final IP destination. An attempt was made to find a route to forward the input datagrams to their final destination.

**Table 9-29. IP Group (cont.)**

Member	Access	Description
ipInUnknownProtos	READ-ONLY	The number of locally-addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.
ipInDiscards	READ-ONLY	The number of input IP datagrams which were discarded. No problems were encountered to prevent their continued processing.
ipInDelivers	READ-ONLY	The total number of input datagrams successfully delivered to IP user-protocols.
ipOutRequests	READ-ONLY	The total number of IP datagrams that local IP user-protocols supplied to IP in requests for transmission.
ipOutDiscards	READ-ONLY	The number of output IP datagrams which were discarded. No problem was encountered to prevent transmission of these datagrams to their destination.
ipOutNoRoutes	READ-ONLY	The number of IP datagrams discarded because no route could be found to transmit them to their destination.
ipReasmTimeout	READ-ONLY	The maximum number of seconds that received fragments are held while they are awaiting reassembly at this entity.
ipReasmReqds	READ-ONLY	The number of IP fragments received which needed to be reassembled at this entity.
ipReasmOKs	READ-ONLY	The number of IP datagrams successfully re-assembled.
ipReasmFails	READ-ONLY	The number of failures detected by the IP re-assembly algorithm.
ipFragOKs	READ-ONLY	The number of IP datagrams that have been successfully fragmented at this entity.
ipFragFails	READ-ONLY	The number of IP datagrams discarded because they needed to, but could not, be fragmented at this entity.
ipFragCreates	READ-ONLY	The number of IP datagram fragments that have been generated as a result of fragmentation at this entity.
ipAddrTable	NOT-ACCESSIBLE	The table of addressing information relevant to this entity's IP addresses.

**Table 9-29. IP Group (cont.)**

Member	Access	Description
ipAddrEntry	NOT-ACCESSIBLE	The addressing information for one of this entity's IP addresses.
ipAdEntAddr	READ-ONLY	The IP address to which this entry's addressing information pertains.
ipAdEntIfIndex	READ-ONLY	The index value uniquely identifies the interface to which this entry is applicable. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
ipAdEntNetMask	READ-ONLY	The subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the hosts bits set to 0.
ipAdEntBcastAddr	READ-ONLY	The value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry.
ipAdEntReasmMaxSize	READ-ONLY	The size of the largest IP datagram this entity can re-assemble from incoming IP-fragmented datagrams received on this interface.
ipRoutingDiscards	READ-ONLY	The number of routing entries that were chosen to be discarded even though they are valid.

## The ipRouteTable

The IP Routing table contains an entry for each route presently known to this entity. This table is a copy of the Nucleus NET Routing table.

**Table 9-30. ipRouteTable**

Member	Access	Description
ipRouteTable	NOT-ACCESSIBLE	This entity's IP Routing table.
ipRouteEntry	NOT-ACCESSIBLE	A route to a particular destination.
ipRouteDest	READ-WRITE	The destination IP address of this route.



**Table 9-30. ipRouteTable (cont.)**

Member	Access	Description
ipRouteIfIndex	READ-WRITE	The index value uniquely identifies the local interface through which the next hop of this route should be reached. The interface identified by the value of this index is the same interface identified using the same value for ifIndex.
ipRouteMetric1	READ-WRITE	The primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
ipRouteMetric2	READ-WRITE	An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
ipRouteMetric3	READ-WRITE	An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
ipRouteMetric4	READ-WRITE	An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.
ipRouteNextHop	READ-WRITE	The IP address of the next hop of this route.
ipRouteType	READ-WRITE	The type of route.
ipRouteProto	READ-ONLY	The routing mechanism via which this route was learned.
ipRouteAge	READ-WRITE	The number of seconds since this route was last updated or otherwise determined to be correct.
ipRouteMask	READ-WRITE	The subnet mask associated with the IP address of this entry.
ipRouteMetric5	READ-WRITE	An alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value should be set to -1.

**Table 9-30. ipRouteTable (cont.)**

Member	Access	Description
ipRouteInfo	READ-ONLY	A reference to MIB definitions specific to the routing protocol responsible for this route, as determined by the route's ipRouteProto value. If this information is not present, its value should be set to the OBJECT IDENTIFIER {0 0}, a syntactically valid object identifier.

## The ipNetToMediaTable

The ipNetToMediaTable contains the entries in the Nucleus NET ARP Cache.

**Table 9-31. ipNetToMediaTable**

Member	Access	Description
ipNetToMediaTable	NOT-ACCESSIBLE	The IP Address Translation table used for mapping from IP addresses to physical addresses.
ipNetToMediaEntry	NOT-ACCESSIBLE	Each entry contains one IP address to 'physical' address equivalence.
ipNetToMediaIfIndex	READ-WRITE	The interface on which this entry's equivalence is effective. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.
ipNetToMediaPhysAddress	READ-WRITE	The media-dependent 'physical' address.
ipNetToMediaNetAddress	READ-WRITE	The IP address corresponding to the media-dependent 'physical' address.
ipNetToMediaType	READ-WRITE	The type of mapping.

## The ICMP Group

The ICMP Group contains statistics about ICMP.

**Table 9-32. ICMP Group**

Member	Access	Description
icmpInMsgs	READ-ONLY	The total number of ICMP messages that the entity received.
icmpInErrors	READ-ONLY	The number of ICMP messages that the entity received but determined as having ICMP-specific errors.
icmpInDestUnreachs	READ-ONLY	The number of ICMP Destination Unreachable messages received.
icmpInTimeExcds	READ-ONLY	The number of ICMP Time Exceeded messages received.
icmpInParmProbs	READ-ONLY	The number of ICMP Parameter Problem messages received.
icmpInSrcQuenchs	READ-ONLY	The number of ICMP Source Quench messages received.
icmpInRedirects	READ-ONLY	The number of ICMP Redirect messages received.
icmpInEchos	READ-ONLY	The number of ICMP Echo (request) messages received.
icmpInEchoReps	READ-ONLY	The number of ICMP Echo Reply messages received.
icmpInTimestamps	READ-ONLY	The number of ICMP Timestamp (request) messages received.
icmpInTimestampReps	READ-ONLY	The number of ICMP Timestamp Reply messages received.
icmpInAddrMasks	READ-ONLY	The number of ICMP Address Mask Request messages received.
icmpInAddrMaskReps	READ-ONLY	The number of ICMP Address Mask Reply messages received.
icmpOutMsgs	READ-ONLY	The total number of ICMP messages which this entity attempted to send.
icmpOutErrors	READ-ONLY	The number of ICMP messages that this entity did not send due to problems discovered within ICMP.

**Table 9-32. ICMP Group (cont.)**

Member	Access	Description
icmpOutDestUnreachs	READ-ONLY	The number of ICMP Destination Unreachable messages sent.
icmpOutTimeExcds	READ-ONLY	The number of ICMP Time Exceeded messages sent.
icmpOutParmProbs	READ-ONLY	The number of ICMP Parameter Problem messages sent.
icmpOutSrcQuenchs	READ-ONLY	The number of ICMP Source Quench messages sent.
icmpOutRedirects	READ-ONLY	The number of ICMP Redirect messages sent.
icmpOutEchos	READ-ONLY	The number of ICMP Echo (request) messages sent.
icmpOutEchoReps	READ-ONLY	The number of ICMP Echo Reply messages sent.
icmpOutTimestamps	READ-ONLY	The number of ICMP Timestamp (request) messages sent.
icmpOutTimestampReps	READ-ONLY	The number of ICMP Timestamp Reply messages sent.
icmpOutAddrMasks	READ-ONLY	The number of ICMP Address Mask Request messages sent.
icmpOutAddrMaskReps	READ-ONLY	The number of ICMP Address Mask Reply messages sent.

## The TCP Group

The TCP Group contains information about TCP statistics and connections.

**Table 9-33. TCP Group**

Member	Access	Description
tcpRtoAlgorithm	READ-ONLY	The algorithm used to determine the timeout value used for retransmitting unacknowledged octets.
tcpRtoMin	READ-ONLY	The minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds.

**Table 9-33. TCP Group (cont.)**

Member	Access	Description
tcpRtoMax	READ-ONLY	The maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds.
tcpMaxConn	READ-ONLY	The limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value -1.
tcpActiveOpens	READ-ONLY	The number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.
tcpPassiveOpens	READ-ONLY	The number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.
tcpAttemptFails	READ-ONLY	The number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.
tcpEstabResets	READ-ONLY	The number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.
tcpCurrEstab	READ-ONLY	The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.
tcpInSegs	READ-ONLY	The total number of segments received, including those received in error.
tcpOutSegs	READ-ONLY	The total number of segments sent, including those on current connections, but excluding those containing only retransmitted octets.
tcpRetransSegs	READ-ONLY	The total number of segments retransmitted.
tcpConnTable	NOT-ACCESSIBLE	A table containing TCP connection-specific information.
tcpConnEntry	NOT-ACCESSIBLE	Information about a particular current TCP connection.
tcpConnState	READ-WRITE	The state of this TCP connection.

**Table 9-33. TCP Group (cont.)**

Member	Access	Description
tcpConnLocalAddress	READ-ONLY	The local IP address for this TCP connection. In the case of a connection in the listen state which is willing to accept connections for any IP interface associated with the node, the value 0.0.0.0 is used.
tcpConnLocalPort	READ-ONLY	The local port number for this TCP connection.
tcpConnRemAddress	READ-ONLY	The remote IP address for this TCP connection.
tcpConnRemPort	READ-ONLY	The remote port number for this TCP connection.
tcpInErrs	READ-ONLY	The total number of segments received in error.
tcpOutRsts	READ-ONLY	The number of TCP segments sent that contain the RST flag.

## The UDP Group

The UDP Group contains the information about UDP statistics and connections.

**Table 9-34. UDP Group**

Member	Access	Description
udpInDatagrams	READ-ONLY	The total number of UDP datagrams delivered to UDP users.
udpNoPorts	READ-ONLY	The total number of received UDP datagrams for which there was no application at the destination port.
udpInErrors	READ-ONLY	The number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.
udpOutDatagrams	READ-ONLY	The total number of UDP datagrams sent from this entity.
udpTable	NOT-ACCESSIBLE	A table containing UDP listener information.
udpEntry	NOT-ACCESSIBLE	Information about a particular current UDP listener.
udpLocalAddress	READ-ONLY	The local IP address for this UDP listener. In the case of a UDP listener, which is willing to accept datagrams for any IP interface associated with the node, the value 0.0.0.0 is used.
udpLocalPort	READ-ONLY	The local port number for this UDP listener.

## The EGP Group

The EGP group is mandatory for all systems that implement EGP. Since Nucleus NET does not implement EGP, this group is turned off by default and excluded from the Nucleus SNMP build.

**Table 9-35. EGP Group**

Member	Access	Description
egpInMsgs	READ-ONLY	The number of EGP messages received without error.
egpInErrors	READ-ONLY	The number of EGP messages received that proved to be in error.
egpOutMsgs	READ-ONLY	The total number of locally generated EGP messages.
egpNeighTable	NOT-ACCESSIBLE	The EGP neighbor table.
egpNeighEntry	NOT-ACCESSIBLE	Information about this entity's relationship with a particular EGP neighbor.
egpNeighState	READ-ONLY	The EGP state of the local system with respect to this entry's EGP neighbor.
egpNeighAddr	READ-ONLY	The IP address of this entry's EGP neighbor.
egpNeighAs	READ-ONLY	The autonomous system of this EGP peer. Zero should be specified if the autonomous system number of the neighbor is not yet known.
egpNeighInMsgs	READ-ONLY	The number of EGP messages received without error from this EGP peer.
egpNeighInErrs	READ-ONLY	The number of EGP messages received from this EGP peer that proved to be in error.
egpNeighOutMsgs	READ-ONLY	The number of locally generated EGP messages to this EGP peer.
egpNeighOutErrs	READ-ONLY	The number of locally generated EGP messages not sent to this EGP peer due to resource limitations within an EGP entity.
egpNeighInErrMsgs	READ-ONLY	The number of EGP-defined error messages received from this EGP peer.
egpNeighOutErrMsgs	READ-ONLY	The number of EGP-defined error messages sent to this EGP peer.
egpNeighStateUps	READ-ONLY	The number of EGP state transitions to the UP state with this EGP peer.

**Table 9-35. EGP Group (cont.)**

Member	Access	Description
egpNeighStateDowns	READ-ONLY	The number of EGP state transitions from the UP state to any other state with this EGP peer.
egpNeighIntervalHello	READ-ONLY	The interval between EGP Hello command retransmissions (in hundredths of a second).
egpNeighIntervalPoll	READ-ONLY	The interval between EGP poll command retransmissions (in hundredths of a second).
egpNeighMode	READ-ONLY	The polling mode of this EGP entity, either passive or active.
egpNeighEventTrigger	READ-WRITE	A control variable used to trigger operator-initiated Start and Stop events.
egpAs	READ-ONLY	The autonomous system number of this EGP entity.

## The SNMP Group

The SNMP Group contains information about SNMP Engine statistics.

**Table 9-36. SNMP Group**

Member	Access	Description
snmpInPkts	READ-ONLY	The total number of messages delivered to the Nucleus SNMP entity from the transport service.
snmpOutPkts	READ-ONLY	The total number of Nucleus SNMP Messages that were passed from the Nucleus SNMP protocol entity to the transport service.
snmpInBadVersions	READ-ONLY	The total number of SNMP Messages that were delivered to the Nucleus SNMP protocol entity with an unsupported Nucleus SNMP version.
snmpInBadCommunityNames	READ-ONLY	The total number of SNMP Messages delivered to the Nucleus SNMP protocol entity that used a SNMP community name not known to the entity.
snmpInBadCommunityUses	READ-ONLY	The total number of SNMP Messages delivered to the SNMP protocol entity representing an SNMP operation not allowed by the Nucleus SNMP community named in the Message.



**Table 9-36. SNMP Group (cont.)**

Member	Access	Description
snmpInASNParseErrs	READ-ONLY	The total number of ASN.1 or BER errors encountered by the Nucleus SNMP protocol entity when decoding received SNMP Messages.
snmpInTooBigs	READ-ONLY	The total number of SNMP PDUs that were delivered to the Nucleus SNMP protocol entity and for which the value of the error-status field is 'tooBig'.
snmpInNoSuchNames	READ-ONLY	The total number of SNMP PDUs that were delivered to the Nucleus SNMP protocol entity and for which the value of the error-status field is 'noSuchName'.
snmpInBadValues	READ-ONLY	The total number of SNMP PDUs that were delivered to the Nucleus SNMP protocol entity and for which the value of the error-status field is 'badValue'.
snmpInReadOnlys	READ-ONLY	The total number of valid SNMP PDUs that were delivered to the Nucleus SNMP protocol entity and for which the value of the error-status field is 'readOnly'.
snmpInGenErrs	READ-ONLY	The total number of SNMP PDUs that were delivered to the Nucleus SNMP protocol entity and for which the value of the error-status field is 'genErr'.
snmpInTotalReqVars	READ-ONLY	The total number of MIB objects which have been retrieved successfully by the Nucleus SNMP protocol entity as the result of receiving valid SNMP Get-Request and Get-Next PDUs.
snmpInTotalSetVars	READ-ONLY	The total number of MIB objects which have been altered successfully by the Nucleus SNMP protocol entity as the result of receiving valid SNMP Set-Request PDUs.
snmpInGetRequests	READ-ONLY	The total number of SNMP Get-Request PDUs that have been accepted and processed by the Nucleus SNMP protocol entity.

**Table 9-36. SNMP Group (cont.)**

Member	Access	Description
snmpInGetNexts	READ-ONLY	The total number of SNMP Get-Next PDUs that have been accepted and processed by the Nucleus SNMP protocol entity.
snmpInSetRequests	READ-ONLY	The total number of SNMP Set-Request PDUs that have been accepted and processed by the Nucleus SNMP protocol entity.
snmpInGetResponses	READ-ONLY	The total number of SNMP Get-Response PDUs that have been accepted and processed by the Nucleus SNMP protocol entity.
snmpInTraps	READ-ONLY	The total number of SNMP Trap PDUs that have been accepted and processed by the Nucleus SNMP protocol entity.
snmpOutTooBigs	READ-ONLY	The total number of SNMP PDUs that were generated by the Nucleus SNMP protocol entity and for which the value of the error-status field is 'tooBig'.
snmpOutNoSuchNames	READ-ONLY	The total number of SNMP PDUs that were generated by the Nucleus SNMP protocol entity and for which the value of the error-status is 'noSuchName'.
snmpOutBadValues	READ-ONLY	The total number of SNMP PDUs that were generated by the Nucleus SNMP protocol entity and for which the value of the error-status field is 'badValue'.
snmpOutGenErrs	READ-ONLY	The total number of SNMP PDUs that were generated by the Nucleus SNMP protocol entity and for which the value of the error-status field is 'genErr'.
snmpOutGetRequests	READ-ONLY	The total number of SNMP Get-Request PDUs that have been generated by the Nucleus SNMP protocol entity.
snmpOutGetNexts	READ-ONLY	The total number of SNMP Get-Next PDUs that have been generated by the Nucleus SNMP protocol entity.
snmpOutSetRequests	READ-ONLY	The total number of SNMP Set-Request PDUs that have been generated by the Nucleus SNMP protocol entity.

Table 9-36. SNMP Group (cont.)

Member	Access	Description
snmpOutGetResponses	READ-ONLY	The total number of SNMP Get-Response PDUs that have been generated by the Nucleus SNMP protocol entity.
snmpOutTraps	READ-ONLY	The total number of Nucleus SNMP Trap PDUs that have been generated by the SNMP protocol entity.
snmpEnableAuthenTraps	READ-WRITE	Indicates whether the Nucleus SNMP agent process is permitted to generate authentication-failure traps.

## Nucleus SNMPv3 MIB

The Management Information Base for the Nucleus SNMP (SNMPv3 MIB) is built into the Nucleus SNMP library by default. You have the option to remove particular components from the build.

## Nucleus SNMPv3 MIB: Defines

### snmp\_cfg.h

The user-definable region for the Nucleus SNMPv3 MIB is noted in this file. Although you may define other variables, normally only those within this definable region are required.

The following defines enable you to exclude Nucleus SNMPv3 MIB components from Nucleus SNMP.

### INCLUDE\_MIB\_USM

Setting this macro to `NU_TRUE` includes the USM (User-based Security Model) MIB component in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 for information on the User-based Security Model.

### INCLUDE\_MIB\_VACM

Setting this macro to `NU_TRUE` includes the VACM (View-based Access Control Model) MIB component in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 for information on the View-based Access Control Model.

## INCLUDE\_MIB\_SNMP\_ENGINE

Setting this macro to NU\_TRUE includes the SNMP Engine MIB in the build.

## INCLUDE\_MIB\_CBSM

Setting this macro to NU\_TRUE includes the Community Based Security Model (CBSM) MIB component in the build. Refer to [“Advanced SNMP Topics”](#) on page 1693 for information on Community Based Security Model.

## INCLUDE\_MIB\_MPD

Setting this macro to NU\_TRUE includes the MPD MIB component in the build.

## INCLUDE\_MIB\_TARGET

Setting this macro to NU\_TRUE includes the target MIB in the build.

## INCLUDE\_MIB\_NO

Setting this macro to NU\_TRUE includes the Notification MIB in the build.

# Nucleus SNMPv3 MIB Managed Objects

This section lists the Nucleus-supported SNMPv3 MIB data objects, as defined by the Internet Society standards document RFC 3584.

## SNMP Engine Group

The Nucleus SNMP Engine Group is defined by the structure SNMP\_ENGINE\_STRUCT in the file *include/networking/snmp.h*. [Table 9-37](#) lists the Nucleus-supported SNMPv3 members for the SNMP Engine Group data object.

**Table 9-37. Nucleus SNMP Engine Group**

Member	Access	Description
snmpEngineID	READ-ONLY	An SNMP Engine’s administratively-unique identifier.
snmpEngineBoots	READ-ONLY	The number of times that the SNMP Engine has (re-)initialized itself since snmpEngineID was last configured.
snmpEngineTime	READ-ONLY	The number of seconds since the value of the snmpEngineBoots object last changed.

**Table 9-37. Nucleus SNMP Engine Group (cont.)**

Member	Access	Description
snmpEngineMaxMessageSize	READ-ONLY	The maximum length, in octets, of an SNMP message this SNMP Engine can send or receive and process. Maximum length is determined as the minimum of the maximum message size values supported among all of the transports available to, and supported by, the Engine.

## SNMP MPD Stats Group

The Nucleus SNMP MPD Group is defined by the structure `SNMP_MPD_STRUCT` in the file `include/networking/snmp_mp.h`. [Table 9-38](#) lists the Nucleus-supported SNMPv3 members for the SNMP MPD Stats Group data object.

**Table 9-38. Nucleus SNMP MPD Stats Group**

Member	Access	Description
snmpUnknownSecurityModels	READ-ONLY	The total number of packets received by the SNMP Engine but dropped because they referenced a securityModel either not known or not supported by the SNMP Engine.
snmpInvalidMsgs	READ-ONLY	The total number of packets received by the SNMP Engine which were dropped because there were invalid or inconsistent components in the SNMP message.
snmpUnknownPDUHandlers	READ-ONLY	The total number of packets received by the SNMP Engine but dropped because the PDU contained in the packet could not be passed to an application responsible for handling the pduType. For example, no SNMP application had registered for the proper combination of the context EngineID and the pduType.

## SNMP Target Address Table

The Nucleus SNMP Target Address table is defined by the structure [SNMP\\_TARGET\\_ADDRESS\\_TABLE](#) in the file *os/include/networking/tgr\_mib.h*. [Table 9-39](#) lists the Nucleus-supported SNMPv3 members for the SNMP Target Address Table data object.

**Table 9-39. Nucleus SNMP Target Address Table**

Member	Access	Description
snmpTargetAddrName	NOT-ACCESSIBLE	The locally arbitrary, but unique identifier associated with this snmpTargetAddrEntry.
snmpTargetAddrTDomain	READ-ONLY	This object indicates the transport type of the address contained in the snmpTargetAddrTAddress object.
snmpTargetAddrTAddress	READ-ONLY	This object contains a transport address. The format of this address depends on the value of the snmpTargetAddrTDomain object.
snmpTargetAddrTimeout	READ-ONLY	This object should reflect the expected maximum round trip time for communicating with the transport address defined by this row. When a message is sent to this address, and a response (if one is expected) is not received within this time period, implementation may assume that the response will not be delivered.
snmpTargetAddrRetryCount	READ-ONLY	This object specifies a default number of retries to be attempted when a response is not received for a generated message. An application may provide its own retry count, in which case the value of this object is ignored.
snmpTargetAddrTagList	READ-ONLY	This object contains a list of tag values which are used to select target addresses for a particular operation.

**Table 9-39. Nucleus SNMP Target Address Table (cont.)**

Member	Access	Description
snmpTargetAddrParams	READ-ONLY	The value of this object identifies an entry in the snmpTargetParamsTable. The identified entry contains SNMP parameters to use when generating messages to be sent to this transport address.
snmpTargetAddrStorageType	READ-ONLY	The storage type for this conceptual row.
snmpTargetAddrRowStatus	READ-ONLY	The status of this conceptual row.

## SNMP Target Params Table

The Nucleus SNMP Target Params table is defined by the structure [SNMP\\_TARGET\\_PARAMS\\_TABLE](#) in the file *include/networking/tgr\_mib.h*. [Table 9-40](#) lists the Nucleus-supported SNMPv3 members for the SNMP Target Params Table data object.

**Table 9-40. Nucleus SNMP Target Params Table**

Member	Access	Description
snmpTargetParamsName	NOT-ACCESSIBLE	The locally arbitrary, but unique identifier associated with this snmpTargetParams Entry.
snmpTargetParamsMPModel	READ-ONLY	The Message Processing Model to be used when generating Nucleus SNMP messages using this entry.
snmpTargetParamsSecurityModel	READ-ONLY	The Security Model to be used when generating Nucleus SNMP messages using this entry.
snmpTargetParamsSecurityName	READ-ONLY	The securityName which identifies the principal on whose behalf SNMP messages are generated using this entry.
snmpTargetParamsSecurityLevel	READ-ONLY	The level of security to be used when generating SNMP messages using this entry.
snmpTargetParamsStorageType	READ-ONLY	The storage type for this conceptual row.
snmpTargetParamsRowStatus	READ-ONLY	The status of this conceptual row.

## SNMP Notify Table

The Nucleus SNMP Notify table is defined by the structure [SNMP\\_NOTIFY\\_TABLE](#) in the file *include/networking/snmp\_no.h*. [Table 9-41](#) lists the Nucleus-supported SNMPv3 members for the SNMP Notify Table data object.

**Table 9-41. Nucleus SNMP Notify Table**

Member	Access	Description
snmpNotifyName	NOT-ACCESSIBLE	The locally arbitrary, but unique identifier associated with this snmpNotifyEntry.
snmpNotifyTag	READ-ONLY	This object contains a single tag value which is used to select entries in the snmpTargetAddrTable. If this object contains a value of zero length, no entries are selected.
snmpNotifyType	READ-ONLY	This object determines the type of notification to be generated for entries in the snmpTargetAddrTable selected by the corresponding instance of snmpNotifyTag. This value is only used when generating notifications, and is ignored when using the snmpTargetAddrTable for other purposes.
snmpNotifyStorageType	READ-ONLY	The storage type for this conceptual row.
snmpNotifyRowStatus	READ-ONLY	The status of this conceptual row.

## SNMP Notify Filter Profile Table

The Nucleus SNMP Notify Filter Profile table is defined by the structure [SNMP\\_NOTIFY\\_FILTER\\_PROFILE\\_TABLE](#) in the file *include/networking/snmp\_no.h*. [Table 9-42](#) lists the Nucleus-supported SNMPv3 members for the SNMP Notify Filter Profile Table data object.

**Table 9-42. Nucleus SNMP Notify Filter Profile Table**

Member	Access	Description
snmpNotifyFilterProfileName	READ-ONLY	The name of the Notify Filter Profile to be used when generating notifications using the corresponding entry in the snmp TargetAddrTable.
snmpNotifyFilterProfileStorType	READ-ONLY	The storage type of this conceptual row.
snmpNotifyFilterProfileRowStatus	READ-ONLY	The status of this conceptual row.



## SNMP Notify Filter Table

The Nucleus SNMP Notify Filter table is defined by the structure [SNMP\\_NOTIFY\\_FILTER\\_TABLE](#) in the file *include/networking/snmp\_no.h*. [Table 9-43](#) lists the Nucleus-supported SNMPv3 members for the SNMP Notify Filter Table data object.

**Table 9-43. Nucleus SNMP Notify Filter Table**

Member	Access	Description
snmpNotifyFilterSubtree	NOT-ACCESSIBLE	The MIB subtree which, when combined with the corresponding instance of snmp NotifyFilterMask, defines a family of subtrees which are included in or excluded from the Notify Filter Profile.
snmpNotifyFilterMask	READ-ONLY	The bit mask which, in combination with the corresponding instance of snmpNotify FilterSubtree, defines a family of subtrees which are included in or excluded from the Notify Filter Profile.
snmpNotifyFilterType	READ-ONLY	This object indicates whether the family of filter subtrees defined by this entry are included in or excluded from a filter.
snmpNotifyFilterStorageType	READ-ONLY	The storage type of this conceptual row.
snmpNotifyFilterRowStatus	READ-ONLY	The status of this conceptual row.

## USM User Group

The USM User Group is defined by the structure [USM\\_USERS\\_STRUCT](#) in the file *include/networking/usm.h*. [Table 9-44](#) lists the Nucleus-supported SNMPv3 members for the USM User Group data object.

**Table 9-44. Nucleus USM User Group**

Member	Access	Description
usmUserEngineID	NOT-ACCESSIBLE	An SNMP Engine's administratively-unique identifier. In a simple agent, this value is always that agent's own snmp EngineID value. The value can also take the value of the snmpEngineID of a remote SNMP Engine with which this user can communicate.

**Table 9-44. Nucleus USM User Group (cont.)**

Member	Access	Description
usmUserName	NOT-ACCESSIBLE	A human readable string representing the name of the user. This is the (User-based Security) Model dependent security ID.
usmUserSecurityName	READ-ONLY	A human readable string representing the user in Security Model independent format.
usmUserCloneFrom	READ-CREATE	A pointer to another conceptual row in this usmUserTable. The user in this other conceptual row is called the clone-from user.
usmUserAuthProtocol	READ-CREATE	An indication of whether messages sent on behalf of this user to/from the SNMP Engine identified by usmUserEngineID, can be authenticated, and if so, the type of Authentication Protocol which is used.
usmUserAuthKeyChange	READ-CREATE	When modified, this object causes the secret authentication key to be modified via a one-way function. The key is used for messages sent between this user and the SNMP Engine identified by usmUserEngineID.
usmUserOwnAuthKeyChange	READ-ONLY	Behaves exactly as usmUserAuthKeyChange, with one notable difference: in order for the set operation to succeed, the usmUserName of the operation request must match the usmUserName that indexes the row which is targeted by this operation.
usmUserPrivProtocol	READ-CREATE	An indication of whether messages sent on behalf of this user to/from the SNMP Engine identified by usmUserEngineID, can be protected from disclosure, and if so, the type of Privacy Protocol which is used.

**Table 9-44. Nucleus USM User Group (cont.)**

Member	Access	Description
usmUserPrivKeyChange	READ-CREATE	An object, which when modified, causes the secret encryption key used for messages sent on behalf of this user to/from the SNMP Engine identified by usmUser EngineID, to be modified via a one-way function.
usmUserOwnPrivKeyChange	READ-ONLY	Behaves exactly as usmUserPrivKeyChange, with one notable difference: in order for the Set operation to succeed, the usmUserName of the operation requester must match the usmUserName that indexes the row which is targeted by this operation. In addition, the USM security model must be used for this operation.
usmUserPublic	READ-CREATE	A publicly-readable value which can be written as part of the procedure for changing a user's secret authentication and/or privacy key, and later read to determine whether the change of the secret was effected.
usmUserStorageType	READ-CREATE	The storage type for this conceptual row.
usmUserStatus	READ-CREATE	The status of this conceptual row.

## VACM Context Group

The VACM Context Group is defined by the structure VACM\_CONTEXT\_STRUCT in the file *include/networking/vacm.h*. [Table 9-45](#) lists the Nucleus-supported SNMPv3 members for the Vacm Context Group data object.

**Table 9-45. Nucleus Vacm Context Group**

Member	Access	Description
vacmContextName	READ-ONLY	A human readable name identifying a particular context at a particular SNMP entity.

## VACM Security To Group

The VACM Security To Group is defined by the structure [VACM\\_SEC2GROUP\\_STRUCT](#) in the file *include/networking/vacm.h*. [Table 9-46](#) lists the Nucleus-supported SNMPv3 members for the VACM Security To Group data object.

**Table 9-46. Nucleus Vacm Security to Group**

Member	Access	Description
vacmSecurityModel	NOT-ACCESSIBLE	The Security Model used to reference the vacmSecurityName .
vacmSecurityName	NOT-ACCESSIBLE	The securityName for the principal, represented in a Security Model independent format, which is mapped by this entry to a groupName.
vacmGroupName	READ-CREATE	The name of the group to which this entry (for example, the combination of securityModel and securityName) belongs.
vacmSecurityToGroupStorageType	READ-CREATE	The storage type for this conceptual row.
vacmSecurityToGroupStatus	READ-CREATE	The status of this conceptual row.

## VACM Access Group

The VACM Access Group is defined by the structure [VACM\\_ACCESS\\_STRUCT](#) in the file *include/networking/vacm.h*. [Table 9-47](#) lists the Nucleus-supported SNMPv3 members for the Vacm Access Group data object.

**Table 9-47. Nucleus Vacm Access Group**

Member	Access	Description
vacmAccessContextPrefix	NOT-ACCESSIBLE	In order to gain the access rights allowed by this conceptual row, a contextName must match exactly (if the value of vacmAccess ContextMatch is 'exact') or partially (if the value of vacm AccessContextMatch is 'prefix') to the value of the instance of this object.
vacmAccessSecurityModel	NOT-ACCESSIBLE	In order to gain the access rights allowed by this conceptual row, this securityModel must be in use.

**Table 9-47. Nucleus Vacm Access Group (cont.)**

Member	Access	Description
vacmAccessSecurityLevel	NOT-ACCESSIBLE	The minimum level of security required in order to gain the access rights allowed by this conceptual row. A securityLevel of no AuthNoPriv is less than authNoPriv which is less than authPriv.
vacmAccessContextMatch	READ-CREATE	If the value of this object is exact (1), then all rows where the contextName exactly matches vacmAccessContextPrefix are selected. If the value of the object is prefix (2), then all rows where the context name whose starting octets exactly match vacmAccessContext Prefix are selected.
vacmAccessReadViewName	READ-CREATE	The value of an instance of this object identifies the VACM MIB View of the SNMP context to which this conceptual row authorizes read access.
vacmAccessWriteViewName	READ-CREATE	The value of an instance of this object identifies the VACM MIB View of the SNMP context to which this conceptual row authorizes write access.
vacmAccessNotifyViewName	READ-CREATE	The value of an instance of this object identifies the VACM MIB View of the SNMP context to which this conceptual row authorizes access for notifications.
vacmAccessStorageType	READ-CREATE	The storage type for this conceptual row.
vacmAccessStatus	READ-CREATE	The status of this conceptual row.

## VACM View Tree Family Group

The Vacm View Tree Family Group is defined by the structure [VACM\\_VIEWTREE\\_STRUCT](#) in the file *include/networking/vacm.h*. [Table 9-48](#) lists the Nucleus-supported SNMPv3 members for the Vacm View Tree Family Group data object.

**Table 9-48. Nucleus Vacm View Tree Family Group**

Member	Access	Description
<code>vacmViewTreeFamilyViewName</code>	NOT-ACCESSIBLE	The human readable name for a family of view subtrees.
<code>vacmViewTreeFamilySubtree</code>	NOT-ACCESSIBLE	The MIB subtree which, when combined with the corresponding instance of <code>vacmViewTreeFamilyMask</code> , defines a family of view subtrees.
<code>vacmViewTreeFamilyMask</code>	READ-CREATE	The bit mask which, in combination with the corresponding instance of <code>vacmViewTreeFamilySubtree</code> , defines a family of view subtrees.
<code>vacmViewTreeFamilyType</code>	READ-CREATE	Indicates whether the corresponding instances of <code>vacmViewTreeFamilySubtree</code> and <code>vacmViewTreeFamilyMask</code> define a family of view subtrees which is included in or excluded from the VACM MIB View.
<code>vacmViewTreeFamilyStorageType</code>	READ-CREATE	The storage type for this conceptual row. This is available only if VACM MIB is included in the build.
<code>vacmViewTreeFamilyStatus</code>	READ-CREATE	The status of this conceptual row. This is available only if VACM MIB is included in the build.

## SNMP Community Table

The SNMP Community table is defined by the structure [SNMP\\_NOTIFY\\_TABLE](#) in the file *include/networking/cbsm.h*. [Table 9-49](#) lists the Nucleus-supported SNMPv3 members for the SNMP Community Table data object.

**Table 9-49. Nucleus SNMP Community Table**

Member	Access	Description
snmpCommunityIndex	NOT-ACCESSIBLE	The unique index value of a row in this table.
snmpCommunityName	READ-CREATE	The community string for which a row in this table represents a configuration.
snmpCommunitySecurityName	READ-CREATE	A human readable string representing the corresponding value of snmpCommunity Name in a Security Model independent format.
snmpCommunityContextEngineID	READ-CREATE	The contextEngineID indicating the location of the Context in which management information is accessed when using the community string specified by the corresponding instance of snmp CommunityName.
snmpCommunityContextName	READ-CREATE	The context in which management information is accessed when using the community string specified by the corresponding instance of snmp CommunityName.

**Table 9-49. Nucleus SNMP Community Table (cont.)**

Member	Access	Description
snmpCommunityTransportTag	READ-CREATE	If a management request containing this community is received on a transport endpoint other than the transport endpoints identified by this object, the request is deemed unauthentic. The transports identified by this object are specified in the snmp TargetAddr Table. Entries in the table whose snmp TargetAddrTagList contains this tag value are identified.
snmpCommunityStorageType	READ-CREATE	The storage type for this conceptual row in the snmpCommunityTable. This is available only if VACM MIB is included in the build.
snmpCommunityStatus	READ-CREATE	The status of this conceptual row in the snmpCommunityTable. This is available only if included in the build.

## Nucleus MIB Utility

MIB is defined in an abstract language called Abstract Syntax Notation, version 1 (ASN.1). This language has rules of grammar and notation that require compilation to insure compliance. An MIB represents a database of information that can be retrieved or modified. Nucleus source code for enterprise-specific MIBs can be created using a separate third party MIB Compiler and the Nucleus MIB Utility provided. Detailed instructions for how to use the MIB Utility are provided in the [Generating Source Code Using the MIB Utility](#) section.

For each MIB, a structure is created. For each table in a MIB, another structure is created. An instance of this structure is declared in the main MIB structure. Each MIB also has a .c file and a header (.h) file corresponding to its name. The header file contains the structure corresponding to the MIB, and the .c file contains initializations for this structure. Each table in the MIB also has a .c and a .h file corresponding to its name. The .h file contains the structure definition corresponding to the table. The .c file contains code to add and remove entries from the table.



The code generated by the MIB Utility can be placed into the following two categories.

- Code responsible for maintaining the information corresponding to the MIB

All structures and the corresponding add and remove functions as well as the initialization function for the MIB make up this category.

- Code responsible for handling SNMP requests on MIB.

The .c files for each table in a MIB also contain functions to the Get and Set table entries as well as callback functions for each object. These functions form the second category. This part of the code is responsible for retrieving and setting instances in the MIB. The callback function for an object is invoked whenever that object needs to be set or retrieved.

The implementer of the MIB Engine has the option to use code generated by the MIB Utility or write his own code. In the MGC MIB application note, information about the task was already being maintained in Nucleus PLUS. Therefore, this code was not used.

## Generating Source Code Using the MIB Utility

The Nucleus MIB Utility is a MIB Engine Code Generator that compiles and generates Nucleus source code using MIB in MOSY format as input. The Nucleus MIB Utility generates ANSI C code sufficient to provide most of the structure required to build a MIB Engine. All that is required for the user after compilation is to fill in the details for the callbacks generated. Below are detailed steps for how to use the MIB utility to generate MIB Nucleus source code.

### Prerequisites

- You need an MIB compiler that outputs MOSY format. Mentor Graphics tested the Nucleus MIB Utility with MOSY output from the MG-SOFT and SMICng MIB compilers.

### Procedure

1. Generate the MOSY file from your ASN.1 input file using a third party MIB compiler.
2. Place the generated MIB file, in MOSY format, in the *tools/bin/winnt/x86* directory.
3. Invoke the MIB Utility

To invoke the MIB Utility, open a command window, and change directory to *tools/bin/winnt/x86*. Next, execute the following command:

```
> mc -r {OIDName} {OID} -o {MIBName} {mibfile}.{defs}
```

Alternatively, the utility can be invoked with a command file:

```
> mc -f {inputfile}
```

The Nucleus MIB Utility command options are described in [Table 9-50](#).

**Table 9-50. Nucleus MIB Utility Options**

Option	Description
-r {OIDName} {OID}	This option sets the root OID Name and its corresponding OID. The root OID is the parent of all the OIDs in the MIB file.
-o {MIBName}	This specifies the MIB Name. By default the file name is taken as the MIB Name. If the file name is used at the MIB Name, ensure that the file name does not have any characters that cannot be used in ANSI C variable names (for example hyphens).
{mibfile}.{defs}	The MIB file in MOSY format to be compiled. Ensure that the file name does not have any characters that cannot be used in ANSI C variable names (hyphens, for example).
-f {inputfile}	This takes input from the input file.

## Results

After executing these steps, you have generated a source code folder of MIBName in the directory *tools/bin/winnt/x86*.

## Related Topics

[Nucleus MIB Utility](#)

[Generating Source Code Using the MIB Utility](#)

## Incorporating Generated MIB Source Code into SNMP Agent

Once the source code for the new MIB is generated using the [Nucleus MIB Utility](#), it must be incorporated into the SNMP Agent code in order to be recognized and used by Nucleus SNMP. A specific detailed example of the use of the MIB Utility on the AT MIB is provided in this section.

## Prerequisites

- Follow the steps in [Generating Source Code Using the MIB Utility](#) to generate Nucleus MIB source code.

## Procedure

1. Move the generated enterprise MIB source code folder from *tools/bin/winnt/x86* to the directory and folder in which the application is being built.
2. Modify the Get and Set functions according to the requirement of the MIB. This procedure is demonstrated in the example below:

The source code for the Notifications or Traps is not generated by the MIB Utility. You must write your own code for generating the Notifications or Traps. The following is an example of generating the Notifications or Traps.

```
/* Handle to the notification request structure */
SNMP_NOTIFY_REQ_STRUCT *snmp_notification;

/* OID of notification */
UINT32 notification_oid[] =
    {1, 3, 6, 1, 2, 1, 55, 2, 0, 1};

/* OID of first binded object */
UINT32 binded_object1_oid[] =
    {1, 3, 6, 1, 2, 1, 55, 1, 5, 1, 2, 1, 1};

/* OID of second binded object */
UINT32 binded_object2_oid [] =
    {1, 3, 6, 1, 2, 1, 55, 1, 5, 1, 10};

/* If you successfully got the handle to the notification */
if (SNMP_Get_Notification_Ptr (&snmp_notification) ==
    NU_SUCCESS)
{
    /* Clear out the notification structure. */
    memset (snmp_notification, 0,
        sizeof(SNMP_NOTIFY_REQ_STRUCT));

    /* Update the notification OID. */
    memcpy (snmp_notification->OID.notification_oid,
        notification_oid,
        sizeof(notification_oid));

    /* Update the notification OID length. */
    snmp_notification->OID.oid_len =
        (sizeof(notification_oid) / sizeof(UINT32));

    /* Set first binded object. */
    memcpy (snmp_notification->snmp_object_list[0].Id,
        binded_object1_oid,
        sizeof(binded_object1_oid));

    /* Set OID length. */
    snmp_notification->snmp_object_list[0].IdLen =
        (sizeof(binded_object1_oid) / sizeof(UINT32));

    /* Set second binded object. */
    memcpy (snmp_notification->snmp_object_list[1].Id,
        binded_object2_oid,
```

```
        sizeof(binded_object2_oid));

/* Set OID length. */
snmp_notification->snmp_object_list[1].IdLen =
    (sizeof(binded_object2_oid) / sizeof(UINT32));

/* Update the number of bound objects. */
snmp_notification->snmp_object_list_len = 2;

/* Send the notification. */
SNMP_Notification_Ready (snmp_notification);
}
```

3. Add code to the application using the [SNMP\\_Mib\\_Register](#) so the enterprise MIBs are added at run-time .

## Results

The MIBs are added into Nucleus SNMP.

## Related Topics

[Nucleus MIB Utility](#)

[Generating Source Code Using the MIB Utility](#)

## PPP MIB

If you are using a PPP interface and wish to include the PPP MIB so that link information can be managed remotely, this can be configured in */include/drivers/ppp\_cfg.h* and */include/networking/snmp\_prt.h*. The MIB is defined in RFCs 1471, 1472, and 1473, which collectively makes up the PPP MIB described in this section.

RFC 1471 defines an MIB that includes link statistics counters, status variables, and LCP option configuration. RFC 1472 defines an MIB that manages authentication for each link, and for each PPP user account. RFC 1473 defines an MIB that includes IP information and configuration for each PPP device.

Each of these MIBs is described in detail in the following sections. The actual MIB files are located in */os/networking/snmp/rfc147x*, which is the base directory for the PPP MIB.

## PPP MIB: Managed Objects

To include any of the PPP MIB, the macro `INCLUDE_PPP_MIB` must be defined as `NU_TRUE` in */include/networking/snmp\_prt.h*. This enables the PPP MIB access in the Nucleus SNMP library, but each individual MIB can also be enabled or disabled in */include/drivers/ppp\_cfg.h*, depending on what your application requires. The following sections describe how to enable and disable the available PPP MIBs.

## The PPP Link MIB

The members of the PPP Link MIB consist of information found in the PPP protocol as defined in RFC 1661 and correspond with the LCP layer structures defined in */include/drivers/lcp\_defs.h* and the HDLC layer configuration defined in */include/drivers/hdlc.h*.

To include this MIB, define the macro `INCLUDE_LCP_MIB` to be `NU_TRUE` in *ppp\_cfg.h*.

Agent access functions are located in */os/networking/snmp/rfc147x/src/1471lcp.c*.

MIB access functions are located in */os/drivers/ppp/src/pml.c*.

**Table 9-51. PPP Link MIB**

Member	Access	Description
pppLinkStatusPhysicalIndex	READ-ONLY	The device index of the next lower link layer to which this PPP link receives and transmits packets. For serial links, this is '0', signifying the lowest link layer. For a PPPoE link, this is the index of the Ethernet device to which this PPP link is bound.
pppLinkStatusBadAddresses	READ-ONLY	The number of packets received with an incorrect Address Field.
pppLinkStatusBadControls	READ-ONLY	The number of packets received on this link with an incorrect Control Field.
pppLinkStatusPacketTooLongs	READ-ONLY	The number of received packets that have been discarded because their length exceeded the MRU.
pppLinkStatusBadFCSs	READ-ONLY	The number of received packets that have been discarded due to having an incorrect FCS.
pppLinkStatusLocalMRU	READ-ONLY	The current value of the MRU for the local PPP Entity. This value is the MRU that the remote entity is using when sending packets to the local PPP entity.
pppLinkStatusRemoteMRU	READ-ONLY	The current value of the MRU for the remote PPP Entity. This value is the MRU that the local entity is using when sending packets to the remote PPP entity.
pppLinkStatusLocalToPeerAC CMap	READ-ONLY	The current value of the ACC Map used for sending packets from the local PPP entity to the remote PPP entity.

**Table 9-51. PPP Link MIB (cont.)**

Member	Access	Description
pppLinkStatusPeerToLocalAC CMap	READ- ONLY	The ACC Map used by the remote PPP entity when transmitting packets to the local PPP entity.
pppLinkStatusLocalToRemote ProtocolCompression	READ- ONLY	Indicates whether the local PPP entity uses Protocol Compression when transmitting packets to the remote PPP entity.
PppLinkStatusRemoteToLocal ProtocolCompression	READ- ONLY	Indicates whether the remote PPP entity uses Protocol Compression when transmitting packets to the local PPP entity.
pppLinkStatusLocalToRemote ACCompression	READ- ONLY	Indicates whether the local PPP entity uses Address and Control Compression when transmitting packets to the remote PPP entity.
pppLinkStatusRemoteToLocal ACCompression	READ- ONLY	Indicates whether the remote PPP entity uses Address and Control Compression when transmitting packets to the local PPP entity.
pppLinkStatusTransmitFcsSize	READ- ONLY	The size of the Frame Check Sequence (FCS) in bits that the local node generates when sending packets to the remote node.
pppLinkStatusReceiveFcsSize	READ- ONLY	The size of the Frame Check Sequence (FCS) in bits that the remote node generates when sending packets to the local node.
pppLinkConfigInitialMRU	READ- WRITE	The initial Maximum Receive Unit (MRU) that the local PPP entity advertises to the remote entity. If the value of this variable is '0' then the local PPP entity does not advertise any MRU to the remote entity and the default MRU is assumed. Changing this object takes effect when the link is next restarted.
pppLinkConfigReceiveACCM ap	READ- WRITE	The Asynchronous-Control-Character-Map (ACC) that the local PPP entity requires for use on its receiver side. In effect, this is the ACC Map that is required in order to ensure that the local modem successfully receives all characters.

**Table 9-51. PPP Link MIB (cont.)**

Member	Access	Description
pppLinkConfigTransmitACCM ap	READ- WRITE	The Asynchronous-Control-Character-Map (ACC) that the local PPP entity requires for use on its transmit side.
pppLinkConfigMagicNumber	READ- WRITE	If true (2) then the local node attempts to perform Magic Number negotiation with the remote node. If false (1) then this negotiation is not performed.
pppLinkConfigFcsSize	READ- WRITE	The size of the FCS, in bits, the local node attempts to negotiate for use with the remote node.

## The PPP Security MIB

The PPP Security MIB consists of two tables of authentication information. The Security Config table defines the authentication preferences for a particular link, while the Security Secrets table contains user information. The members of the PPP Security MIB consist of information found in the PAP and CHAP Authentication Protocols as well as User Management information in the UM module of Nucleus NET.

To include this MIB, define the macro `INCLUDE_SEC_MIB` to be true in `/include/drivers/ppp_cfg.h`. Note that `PPP_ENABLE_UM_DATABASE`, located in the same file, also needs to be true if the Security MIB is used.

Agent access functions are located in `/os/networking/snmp/rfc147x/src/1472sec.c`.

MIB access functions are located in `/os/drivers/ppp/src/pmsc.c` and `/os/drivers/ppp/src/pmss.c`.

**Table 9-52. PPP Security MIB**

Member	Access	Description
pppSecurityConfigLink	READ-WRITE	The value of ifIndex that identifies the entry in the interface table that is associated with the local PPP entity's link for which this particular security algorithm is attempted.
pppSecurityConfigPreference	READ-WRITE	The relative preference of the security protocol identified by pppSecurityConfigProtocol. Security protocols with lower values of pppSecurityConfigPreference are tried before protocols with higher values.

**Table 9-52. PPP Security MIB (cont.)**

Member	Access	Description
pppSecurityConfigProtocol	READ-WRITE	Identifies the security protocol to be attempted on the link identified by pppSecurityConfigLink at the preference level identified by pppSecurityConfigPreference.
pppSecurityConfigStatus	READ-WRITE	Setting this object to the value invalid (1) has the effect of invalidating the corresponding entry in the pppSecurityConfigTable. Setting this field to invalid has no effect since PPP devices are static in nature, and thus a security protocol is always associated with it.
pppSecuritySecretsLink	READ-ONLY	The link to which this ID/Secret pair applies. By convention, if the value of this object is 0 then the ID/Secret pair applies to all links.
pppSecuritySecretsIdIndex	READ-ONLY	A unique value for each ID/Secret pair that has been defined for use on this link. This allows multiple ID/Secret pairs to be defined for each link.
pppSecuritySecretsDirection	READ-WRITE	This object defines the direction in which a particular ID/Secret pair is valid. If this object is local-to-remote then the local PPP entity uses the ID/Secret pair when attempting to authenticate the local PPP entity to the remote PPP entity. If this object is remote-to-local then the local PPP entity expects the ID/Secret pair to be used by the remote PPP entity when the remote PPP entity attempts to authenticate itself to the local PPP entity. In Nucleus PPP, setting this variable has no effect since the direction of authentication is defined by the connection function (for example, NU_Dial_PPP_Server) that is used by the application.



**Table 9-52. PPP Security MIB (cont.)**

Member	Access	Description
pppSecuritySecretsProtocol	READ-WRITE	The security protocol (for example, CHAP or PAP) to which this ID/Secret pair applies. Nucleus PPP always uses the protocol of the link that dials out (set using pppSecurityConfigProtocol) or receives a connection, so setting this field has no effect.
pppSecuritySecretsIdentity	READ-WRITE	The Identity (or username) of the ID/Secret pair. Note that setting the username via a remote manager also changes the corresponding entry in the User Management database, and thus affects all permissions for you.
pppSecuritySecretsSecret	READ-WRITE	The secret (password) of the ID/Secret pair. Like the Identity field, setting the password via a remote manager also changes the corresponding entry in the User Management database, and thus affects all permissions for you.
pppSecuritySecretsStatus	READ-WRITE	Setting this object to the value invalid (1) has the effect of invalidating the corresponding entry in the pppSecuritySecretsTable. The table entry is not deleted when marked as invalid, but it is checked when authenticating a remote user, so that an invalid user is not authenticated. Table entries are only deleted by the application using NU_Remove_PPP_User ().

## The PPP NCP MIB

The members of the PPP NCP MIB consist of information found in the IPCP protocol as defined in RFC 1332 and correspond with the NCP layer structures defined in */include/drivers/ncp\_defs.h*. Only IPCP information is covered in this MIB.

To include this MIB, define the macro `INCLUDE_NCP_MIB` to be true in */include/drivers/ppp\_cfg.h*.

Agent access functions are located in */os/networking/snmp/rfc147x/src/1473ncp.c*.

MIB access functions are located in */os/drivers/ppp/src/pmn.c*.

**Table 9-53. PPP NCP MIB**

Member	Access	Description
pppIpOperStatus	READ-ONLY	The operational status of the IP network protocol. If the value of this object is up then the finite state machine for the IP network protocol has reached the Opened state.
pppIpLocalToRemoteCompressionProtocol	READ-ONLY	The IP compression protocol that the local PPP-IP entity uses when sending packets to the remote PPP-IP entity. Since Nucleus PPP does not implement IP compression, this field always has the value 'none'.
pppIpRemoteToLocalCompressionProtocol	READ-ONLY	The IP compression protocol that the remote PPP-IP entity uses when sending packets to the local PPP-IP entity. Since Nucleus PPP does not implement IP compression, this field always has the value 'none'.
pppIpRemoteMaxSlotId	READ-ONLY	The Max-Slot-Id parameter that the remote node has advertised and that is in use on the link. Since Nucleus PPP does not implement IP compression, this field always has the value '0'.
pppIpLocalMaxSlotId	READ-ONLY	The Max-Slot-Id parameter that the local node has advertised and that is in use on the link. Since Nucleus PPP does not implement IP compression, this field always has the value '0'.
pppIpConfigAdminStatus	READ-WRITE	The immediate desired status of the IP network protocol. Setting this object to open injects an administrative open event into the IP network protocol's finite state machine. Setting this object to close injects an administrative close event into the IP network protocol's finite state machine.
pppIpConfigCompression	READ-WRITE	If none (1) then the local node does not attempt to negotiate any IP Compression option. Since Nucleus PPP does not implement IP compression, setting this field is always set successfully, but has no effect internally.

## Advanced SNMP Topics

This chapter includes the extended discussion for advanced users who wish to better understand the internal functionality and configuration of Nucleus SNMP.

## Security Subsystem Configuration

The Security Subsystem (*snmp\_ss.h* and *snmp\_ss.c*) is responsible for Encryption/Decryption as well as the Authentication of SNMP Messages.

Nucleus SNMP implements USM, User-based Security Model (RFC 3414), and CBSM, Community-Based Security Model (version 1 and version 2).

## Adding a New SNMP Security Model

This section describes how to add a new security model and explains the various interfaces of the SNMP security model.

### Procedure

1. Set-up initialization, if required.

On start up, Nucleus SNMP invokes the initialization functions for the Security Models. USM, CBSMv1, and CBSMv2 models use this function to initialize the user database. Initialization is a two-step process, reading from a file and configuring through configuration files. If a real file system is being used, the data is read from file; otherwise, this function fails and Nucleus SNMP enters the configuration mode.

Security Models which require initialization on start up must implement the initialization function with the following prototype (where `MODEL_Init` is the function name):

```
STATUS MODEL_Init (VOID);
```

2. Set-up configuration, if required.

On start up, if initialization fails, Nucleus SNMP goes into configuration mode. The configuration functions are then invoked. USM, CBSMv1, and CBSMv2 use this function to set up initial user accounts.

During configuration mode, Nucleus SNMP configures using the information defined by the *include/networking/snmp\_cfg.h* and *networking/snmp/snmp\_cfg.c* files.

Security Models which require configuration when run for the first time or when initialization fails must implement the configuration function with the following prototype (where `MODEL_Config` is the function name):

```
STATUS MODEL_Config (VOID);
```

3. Implement the secure function, if required.

The secure function for a particular model is invoked when an SNMP message must be encrypted and/or authenticated while sending out a message or notification.

Security models which need to secure messages must implement the secure function with the following prototype (where MODEL\_Secure is the function name):

```
STATUS MODEL_Secure (UINT32 snmp_mp,  
                     UINT8  **whole_message,  
                     UINT32 *msg_len,  
                     UINT32 max_message_size,  
                     UINT32 snmp_sm,  
                     UINT8  *security_engine_id,  
                     UINT32 security_engine_id_len,  
                     UINT8  *security_name,  
                     UINT8  security_level,  
                     UINT8  *scoped_pdu,  
                     VOID    *security_state_ref);
```

- snmp\_mp  
Message Processing Model.
- whole\_message  
Double pointer to the complete message to be sent.
- msg\_len  
Pointer to the Message Length.
- max\_message\_size  
Maximum message size for the Manager.
- snmp\_sm  
Security Model.
- security\_engine\_id  
Pointer to the authoritative Engine ID.
- security\_engine\_id\_len  
Length of the authoritative Engine ID.
- security\_name  
Pointer to the Security Name of the principal.
- security\_level  
Required security level.
- scoped\_pdu

Pointer to the scoped PDU.

- security\_state\_ref

Pointer to the security information.

#### 4. Implement the verify function, if required.

The verify function for a particular model is invoked when an SNMP message is received with the security from this model.

Security Models that need to decrypt/authenticate messages must implement the verify function with the following prototype (where MODEL\_Verify is the function name):

```
STATUS MODEL_Verify (UINT32      snmp_mp,
                     UINT32      max_message_size,
                     UINT8       *security_param,
                     UINT32      snmp_sm,
                     UINT8       *security_level,
                     UINT8       **whole_message,
                     UINT32      *msg_len,
                     UINT8       *security_engine_id,
                     UINT32      *security_engine_id_len,
                     UINT8       *security_name,
                     UINT32      *max_response_pdu,
                     VOID         **security_state_ref,
                     SNMP_ERROR_STRUCT *error_indication)
```

- snmp\_mp

Message Processing Model.

- max\_message\_size

Maximum message size for the manager.

- security\_param

Pointer to the Security parameters.

- snmp\_sm

Security Model.

- security\_level

Pointer to the Security level.

- whole\_message

Double pointer to the whole message as received on the wire.

- msg\_len

Pointer to the Message Length.

- security\_engine\_id

Pointer to the Authoritative Engine ID.

- security\_engine\_id\_len

Pointer to the Authoritative Engine ID length.

- security\_name

Pointer to the Security name of the principal.

- max\_response\_pdu

Pointer to the maximum size for the response scoped PDU.

- security\_state\_ref

Double pointer to the security information for the session.

- error\_indication

Pointer to the status information if any error was encountered.

## Results

The required security is added to your Nucleus SNMP configuration.

## Related Topics

[Security Subsystem Configuration](#)

# SNMP Configurations

This section is intended for the advanced users who wish to add a new Security Model.

## include/networking/snmp\_cfg.h

### SNMP\_SS\_MODELS\_NO

This define sets the number of Security Models in the Agent. To add a new Security Model, simply modify this define and add the required function pointers in `SNMP_Ss_Models` in *networking/snmp/snmp\_cfg.c*. Functions that are not available in the Security Model can be defined as `NU_NULL`. Note that the header file for the model needs to be included in *networking/snmp/snmp\_cfg.c*.

## networking/snmp/snmp\_cfg.c

### SNMP\_Ss\_Models

This global array integrates security models into the agent. To add a new security model, make the following entry in the array:

```
{Model Version, Initialization Function, Configuration Function, Secure  
Function, Verify Function}
```

Functions that are not available in the security model can be defined as NU\_NULL. Note that the header file for the model needs to be included in *networking/snmp/snmp\_cfg.c*.

Snpmp\_Ss\_Models is an array of type [SNMP\\_SS\\_STRUCT](#), defined in [SNMP Data Structures](#).

## User-based Security Model

User-based Security Model (USM) has support for Authentication as well as Privacy.

Nucleus SNMP comes with HMAC-MD5-96 and HMAC-SHA-96 Authentication Protocols as well as CBC-DES Symmetric Encryption protocol for Privacy.

The USM specification encompasses: [Authentication](#), [Timeliness](#), [Privacy](#), [Message Format](#), [Discovery](#), and [Key Management](#).

### Authentication

USM provides data integrity authentication. The message authentication code HMAC, with either hash function MD5 or SHA provides authentication.

### Timeliness

USM protects against message delay or replay.

### Privacy

USM protects against disclosure of message payload. The cipher block chaining (CBC) mode of DES is used for encryption.

### Message Format

USM defines the format which supports functions of authentication, timeliness, and privacy.

## Discovery

USM defines procedures by which one SNMP Engine obtains information about another SNMP Engine.

## Key Management

USM defines procedures for key generation, update, and use.

Like other SNMP security models, USM is required to output the following values upon successful verification. These values are used by the [Access Control Subsystem Configuration](#) for checking access on SNMP-managed objects.

- Context Name – As found in the `usmUserTable` entry used for verification.
- Security name – As passed in the incoming message.
- Security Model – Always USM.
- Security Level – As passed in the incoming message. The valid values are `noAuthNoPriv`, `authNoPriv` and `authPriv`.

For verification purposes, USM has a table, `usmUserTable`, which holds security information about the users.

When run for the first time (or when initialization fails), USM needs to be configured. USM can be configured either by defining the initial values in *`include/networking/snmp_cfg.h`* and *`networking/snmp/snmp_cfg.c`*, or through the serial interface at run-time.

Once configured, users can also be added to USM using the Nucleus SNMP manager. Refer to “[Nucleus SNMPv3 MIB](#)” on page 1669 for more information.

## `include/networking/snmp_cfg.h`

### USM\_PASSWORD\_2\_KEY

SNMP specifications specify a conversion function that converts a USM password into a key which is then used for either authentication or privacy. This conversion is processor intensive and can lead to large initialization times as well as system slow-down on slow processors. By setting this macro to `NU_FALSE`, this conversion is disabled. You are then responsible for passing a key instead of a password to the USM module. A command line utility has been provided which performs the conversion from password to key. This utility can be used to generate the keys. This utility is discussed later in this chapter.



## USM\_USER\_NAME\_INITIAL

This macro defines the user name of the first user. During configuration mode Nucleus SNMP adds a USM user with this name. Nucleus SNMP enters into configuration mode when initialization from files failed. A real file system is required for successful initialization from file. During initialization from files Nucleus SNMP tries to restore its most recent state. If initialization from the file fails, Nucleus SNMP configures itself using the state defined in the configuration files.

## USM\_AUTH\_PASSWORD\_INITIAL

The value of this macro holds the Authentication Password when the USM\_PASSWORD\_2\_KEY macro is defined to NU\_TRUE. Otherwise this macro holds the value of the Authentication Key that can be generated by using the [Password to Key Utility](#) shipped with Nucleus SNMP. This command line utility can be found at *nucleus/os/networking/snmp*.

The user, defined by the macros USM\_USER\_NAME\_INITIAL, USM\_AUTH\_PASSWORD\_INITIAL and USM\_PRIV\_PASSWORD\_INITIAL, uses the HMAC-MD5 Authentication Protocol under default configurations. You can use a different Authentication Protocol by configuring the [Usm\\_User\\_Table](#) in *networking/snmp/snmp\_cfg.c*.

The password must also be entered in the SNMP manager for communication in authenticated mode. Otherwise, the SNMP manager communicates in unauthenticated mode. Refer to the [Access Control Subsystem Configuration](#) section for setting different access rights for different security levels.

## USM\_PRIV\_PASSWORD\_INITIAL

The value of this macro would hold the Privacy Password when the USM\_PASSWORD\_2\_KEY macro is defined to NU\_TRUE. Otherwise, this macro should hold the value of the Privacy Key that can be generated by the [Password to Key Utility](#) shipped with Nucleus SNMP. This command line utility can be found at *nucleus/os/networking/snmp*.

The user, defined by the macros USM\_USER\_NAME\_INITIAL, USM\_AUTH\_PASSWORD\_INITIAL and USM\_PRIV\_PASSWORD\_INITIAL, uses the HMAC-MD5 Authentication Protocol and the CBC-DES privacy protocol under default configurations. You can use a different Authentication and Privacy Protocol by configuring the [Usm\\_User\\_Table](#) in *networking/snmp/snmp\_cfg.c*.

The password must also be entered in the SNMP manager for communication using privacy services. Otherwise, the SNMP manager communicates without using privacy service. The Privacy service can only be used with authentication. Refer to the [Access Control Subsystem Configuration](#) section for setting different access rights for different security levels.

## USM\_MAX\_AUTH\_PROTOCOLS

This macro does not need to be modified unless the Authentication Protocols have been modified in USM. This macro defines the total number of the Authentication Protocols in USM. This value should be at least 1; the first entry is for `usmNoAuthProtocol`. By default, the USM supports `NoAuthProtocol`, `HMAC-MD5` and `HMAC-SHA`.

## USM\_MAX\_PRIV\_PROTOCOLS

This macro does not need to be modified unless the Privacy Protocols have been modified in USM. This macro defines the total number of the Privacy Protocols in USM. This value should be at least one. The first entry is for `usmNoPrivProtocol`. By default, the USM supports `NoPrivacyProtocol` and `CBC-DES`.

## USM\_MAX\_USER\_USERS

This macro defines the total number of the users to be added during the USM configuration. This macro defines the number of entries in the configurable array `Usm_User_Table` in *networking/snmp/snmp\_cfg.c*.

## networking/snmp/snmp\_cfg.c

### Usm\_Auth\_Prot\_Table

This global array integrates the Authentication Protocols in to USM. The USM supports `NoAuthenticationProtocol`, `HMAC-MD5` and `HMAC-SHA` under the default configuration.

---

#### Note



Only advanced users who wish to add or remove the Authentication Protocol should configure this array.

---

To add a new Authentication Protocol, make the following entry into the array:

```
{Protocol Index, Outgoing Processing Function, Incoming Processing  
Function, Key Generation Function, Key Change Function, Authentication  
Parameter's Length, Key Length}
```


Functions that are not available in the protocol can be defined as `NU_NULL`. Note that the header file for the protocol needs to be included in *networking/snmp/snmp\_cfg.c*. Refer to *include/networking/usm.h* for the interfaces for these functions. *src/usm\_md5.c* and *src/usm\_sha.c* provide good examples for the implementers of the new protocols.

`Usm_Auth_Prot_Table` is an array of type `USM_AUTH_PROT_STRUCT`, defined in [SNMP Data Structures](#).

## Usm\_Priv\_Prot\_Table

This global array integrates Privacy Protocols into USM. The USM supports NoPrivacyProtocol and CBC-DES Privacy Protocols under the default configuration.

---

 **Note** Only advanced users who wish to add or remove a Privacy Protocol should configure this array.

---

To add a new Privacy Protocol, make the following entry to the array:

```
{Protocol Index, Outgoing Processing Function, Incoming Processing  
Function, Privacy Parameter's Length}
```

Functions that are not available in the protocol can be defined to NU\_NULL. Note that the header file for the protocol needs to be included in *networking/snmp/snmp\_cfg.c*. Refer to *include/networking/usm.h* for the interfaces for these functions. *src/usm\_des.c* provides a good example for the implementers of the new protocols.

Usm\_Priv\_Prot\_Table is an array of type [USM\\_PRIV\\_PROT\\_STRUCT](#), defined in [SNMP Data Structures](#).

## Usm\_User\_Table

This global array integrates USM users. To add a new user, make the following entry into the array:

```
{User Name, Authentication Protocol Index, Authentication Password,  
Privacy Protocol Index, Privacy Password}
```

Usm\_User\_Table is an array of type [SNMP\\_USM\\_USER\\_STRUCT](#), defined in [SNMP Data Structures](#).

In Nucleus SNMP, the USM users added during configuration use null strings as context names.

## Password to Key Utility

SNMP specifications specify a conversion function that converts a USM password into a key which is then used for either authentication or privacy. This conversion is processor intensive and can lead to large initialization times as well as system slow-down on slow processors. By setting the USM\_PASSWORD\_2\_KEY macro to NU\_FALSE, this conversion is disabled, making the users responsible for passing a key instead of a password to the USM module. A command line utility has been provided to convert from password to key.

To generate an HMAC-MD5 key, the command is “pass2key M <password>”. For example, to generate HMAC-MD5 key for password “authentic”, the following command is used:

```
> pass2key M authentic
```

Password to key conversion:

Algorithm: MD5

Password: authentic

Key: 0x71297942B8DC29A9E6FA2517F3AC2BB6

So, the key to use would be:

```
{ (UINT8) 0x71, (UINT8) 0x29, (UINT8) 0x79, (UINT8) 0x42,
  (UINT8) 0xB8, (UINT8) 0xDC, (UINT8) 0x29, (UINT8) 0xA9,
  (UINT8) 0xE6, (UINT8) 0xFA, (UINT8) 0x25, (UINT8) 0x17,
  (UINT8) 0xF3, (UINT8) 0xAC, (UINT8) 0x2B, (UINT8) 0xB6 }
```

And for password “private”;

```
> pass2key M private
```

Password to key conversion:

Algorithm: MD5

Password: private

Key: 0x3105E284F13100DE0B1EBA82C3F9990A

So, the key to use would be:

```
{ (UINT8) 0x31, (UINT8) 0x05, (UINT8) 0xE2, (UINT8) 0x84,
  (UINT8) 0xF1, (UINT8) 0x31, (UINT8) 0x00, (UINT8) 0xDE,
  (UINT8) 0x0B, (UINT8) 0x1E, (UINT8) 0xBA, (UINT8) 0x82,
  (UINT8) 0xC3, (UINT8) 0xF9, (UINT8) 0x99, (UINT8) 0x0A }
```

To generate an HMAC-SHA key, use the command “pass2key S <password>”. For example, to generate HMAC-MD5 key for password “authentic”, the following command can be used:

```
> pass2key S authentic
```

Password to key conversion:

Algorithm: SHA-1

Password: authentic

Key: 0xA4A5B5DE9177F2D06C666AD3715A0248B4B42989

So, the key to use would be:

```
{ (UINT8) 0xA4, (UINT8) 0xA5, (UINT8) 0xB5, (UINT8) 0xDE,
  (UINT8) 0x91, (UINT8) 0x77, (UINT8) 0xF2, (UINT8) 0xD0,
  (UINT8) 0x6C, (UINT8) 0x66, (UINT8) 0x6A, (UINT8) 0xD3,
  (UINT8) 0x71, (UINT8) 0x5A, (UINT8) 0x02, (UINT8) 0x48,
  (UINT8) 0xB4, (UINT8) 0xB4, (UINT8) 0x29, (UINT8) 0x89 }
```

```
(UINT8) 0x6C, (UINT8) 0x66, (UINT8) 0x6A, (UINT8) 0xD3,  
(UINT8) 0x71, (UINT8) 0x5A, (UINT8) 0x02, (UINT8) 0x48,  
(UINT8) 0xB4, (UINT8) 0xB4, (UINT8) 0x29, (UINT8) 0x89}
```

And for password “private”;

```
> pass2key S private
```

Password to key conversion:

Algorithm: SHA-1

Password: private

Key: 0x4873606A56C22B1EE58304AEE6A377B3AF23FA8

So, the key to use would be:

```
{ (UINT8) 0x48, (UINT8) 0x73, (UINT8) 0x60, (UINT8) 0x6A,  
  (UINT8) 0x56, (UINT8) 0xC2, (UINT8) 0x2B, (UINT8) 0x1E,  
  (UINT8) 0xE5, (UINT8) 0x83, (UINT8) 0x04, (UINT8) 0xAE,  
  (UINT8) 0xEC, (UINT8) 0x6A, (UINT8) 0x37, (UINT8) 0x7B,  
  (UINT8) 0x3A, (UINT8) 0xF2, (UINT8) 0x3F, (UINT8) 0xA8 }
```

---

#### Note



The Privacy Key is also generated using the Authentication Protocol.

---

## Community-Based Security Model

The Community-Based Security Model (CBSM) provides security for SNMPv1 and SNMPv2c message models as defined in RFC 2576. A community is also added for CBSM during configuration.

A community defines an authorized group of people or SNMP Managers for a SNMP Agent. The community name or community string uniquely identifies a community.

CBSM performs the following steps while validating an incoming request. Changing the default community name would be enough security for the average user. For the advanced users who wish to have a better understanding about the relationships of the different tables within Nucleus SNMP, the following information is useful:

- The community string is registered with Nucleus SNMP in the Community table.
- Get the value of Transport Tag from the Community table Entry used.
- If Transport Tag is an empty string then it is ignored for the purpose of matching.
- If Transport Tag is a non-empty string, it searches for an entry in the Transport Address table using the Transport Address (IP address of SNMP manager), the Transport

Domain (domain over which the request was received (currently, Nucleus SNMP only supports UDP)), and the Transport Tag.

- If the Transport Address Entry is found, the authorization is successful; otherwise, it is not successful.

The hosts for a community are added in the [Target Address Table](#).

The CBSM, like other SNMP Security Models, generates the following values that are used by the [Access Control Subsystem](#) for checking access on SNMP managed objects:

- Context Name - As found in the selected entry of the [CBSM\\_Community\\_Table](#).
- Security Name - As found in the selected entry of the [CBSM\\_Community\\_Table](#).
- Security Model - 1 (SNMP\_CBSM\_V1) for SNMPv1 and 2 (SNMP\_CBSM\_V2) for SNMPv2c.
- Security Level - Always NoAuthNoPriv.

## [include/networking/snmp\\_cfg.h](#)

### [CBSM\\_MAX\\_COMMUNITIES](#)

This macro defines the total number of communities to be added during configuration. This macro defines the number of elements in the configurable [CBSM\\_Community\\_Table](#) array in the *networking/snmp/snmp\_cfg.c* file

## [networking/snmp/snmp\\_cfg.c](#)

### [CBSM\\_Community\\_Table](#)

This global array holds the community entries to be configured during configuration mode. To add a new community, make the following entry into the array:

```
{Community Name, Security Name, Group Name, Context Name}
```

For example:

```
{"public", "public", "group1", ""}
```

[CBSM\\_Community\\_Table](#) is an array of type [SNMP\\_CBSM\\_COMMUNITY\\_STRUCT](#), defined in [SNMP Data Structures](#).

## Application Subsystem

The Application Subsystem consists of [Protocol Operations](#), which processes requests by the SNMP manager, and the [Access Control Subsystem Configuration](#) that controls access to the managed objects. Refer to RFC 1905 for the specifications for the protocol operations and RFC 3415 for the specifications of the View-based Access Control Model.

## Protocol Operations

The following information is only for the reader who wishes to better understand the underlying workings of Nucleus SNMP. This section is helpful for the MIB implementer.

The requests by an SNMP manager can be generalized into two categories of [Protocol Operations](#):

- Get operations - enable a manager to retrieve information from the Agent
- Set operations - add information to the agent.

### Get

The Get operation retrieves the value for the instance specified in the request ID. After the message which contains a Get request has been decoded, the request is sent to the Command Responder Application's queue. The Command Responder retrieves the request and calls `MibRequest` if the request is a version 1 PDU and `MIB_V2_Request` if the request is a version 2 PDU. [Figure 9-6](#) shows the processing for both versions.

**Figure 9-6. Get**

Activity	File	Function
<b>Version 1</b>		
For each object in the list	<code>mib.c</code>	<code>MibRequest</code>
Process the request on single object		<code>Request</code>
Get the instance		<code>MIB_Get_Request</code>
Check whether user has required access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Search MIB root table for object		
Invoke the callback function for object		
<b>Version 2</b>		
For each object in the list	<code>mib.c</code>	<code>MIB_V2_Request</code>
Process the request on single object		<code>Request</code>
Get the instance		<code>MIB_Get_Instance</code>
Check whether user has required access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Search MIB root table for object		
Invoke the callback function for object		

## Get-Next

The Get-Next operation retrieves the value for the instance that is “next” to the object-id passed in the request. After the message which contains a Get-Next request has been decoded, the request is sent to the Command Responder Application’s queue. The Command Responder retrieves the request and calls `MibRequest` if the request is a version 1 PDU and `MIB_V2_Request` if the request is a version 2 PDU. [Figure 9-7](#) shows the processing for both versions:

**Figure 9-7. Get-Next**

Activity	File	Function
<b>Version 1</b>		
For each object in the list	<code>mib.c</code>	<code>MibRequest</code>
Process the request on each object		<code>Request</code>
Get next instance		<code>MIB_Get_Next_Instance</code>
Invoke callback function		
Check access of retrieved object	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
If access fails, get next instance		
<b>Version 2</b>		
For each object in the list	<code>mib.c</code>	<code>MIB_V2_Request</code>
Process the request on each object		<code>Request</code>
Get next instance		<code>MIB_Get_Next_Instance</code>
Invoke callback function		
Check access of the retrieved object	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
If access fails, get next instance		

## Get-Bulk

The Get-Bulk operation enables the retrieval of large amounts of data. After the message which contains a Get-Bulk request has been decoded, the request is sent to the Command Responder Application’s queue. The Command Responder retrieves the request and calls `MIB_V2_Request`. Get-Bulk is not supported in version 1 PDUs. [Figure 9-8](#) shows the processing of Get-Bulk:

**Figure 9-8. Get-Bulk**

Activity	File	Function
<b>Version 2</b>		
Process SNMP Request	<code>mib.c</code>	<code>MIB_V2_Request</code>
Process the Get-Bulk request		<code>MIB_Process_Bulk</code>
For each non-repeater		
Get the next instance		<code>Request</code>
For each repeater		
Get the next object		
Invoke callback function of MIB Object		
Check access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Repeat if more instances required		



## Set

The Set operation executes in three phases. In the first phase, the instance(s) are actually set. In the second phase, the values are verified, and if the set was successful, then the data is committed (possibly saved to storage device). If an error is encountered during the first two phases, the third phase is executed. In this phase the previous values are restored. After the message which contains a Set request has been decoded, the request is sent to the Command Responder Application's queue. The Command Responder retrieves the request and calls `MibRequest` if the request is a version 1 PDU and `MIB_V2_Request` if the request is a version 2 PDU. [Figure 9-9](#) and [Figure 9-10](#) show the processing for both versions:

**Figure 9-9. Set Version One**

Activity	File	Function
<b>Version 1</b>		
If request type is SET PDU	<code>mib.c</code>	<code>MibRequest</code>
Set the new value		<code>MIB_Set_Request</code>
For each object		
Get back-up the object's value.		<code>Request</code>
Set the value to the object		<code>Request</code>
Process the Set request		<code>MIB_Set_Instance</code>
Check access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Invoke callback function for object		
If sets were successful, perform commit		
For each instance in the list	<code>mib.c</code>	<code>MIB_Set_Request</code>
Commit the new value		<code>Request</code>
Check access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Invoke callback function for object		
If an error occurred. Perform undo.		
For each instance in the list	<code>mib.c</code>	<code>MIB_Set_Request</code>
Set the old value		<code>Request</code>
Check access	<code>vacm.c</code>	<code>VACM_CheckInstance_Access</code>
Invoke callback function for object		

**Figure 9-10. Set Version Two**

```
Version 2
If request type is SET PDU                mib.c      MIB_V2_Request
Set the new value                          MIB_Set_Request
For each object
    Get back-up the object's value.        Request
    Set the value to the object             Request
    Process the Set request                 MIB_Set_Instance
    Check access                           vacm.c      VACM_CheckInstance_Access
    Invoke callback function for object

If sets were successful, perform commit
For each instance in the list              mib.c      MIB_Set_Request
    Commit the new value                    Request
    Check access                           vacm.c      VACM_CheckInstance_Access
    Invoke callback function for object

If an error occurred. Perform undo.
For each instance in the list              mib.c      MIB_Set_Request
    Set the old value                       Request
    Check access                           vacm.c      VACM_CheckInstance_Access
    Invoke callback function for object
```

## Access Control Subsystem Configuration

The Access Control Subsystem has the responsibility for checking whether a specific type of access (read, write, notify) to a particular Managed MIB object (instance) is allowed. Nucleus SNMP implements the VACM, as defined in RFC 3415.

Access control occurs in the SNMP entity when processing SNMP retrieval (read) or modification (set) request messages. For example, the Command Responder Application applies Access Control when processing requests that it received from a Command Generator Application. These requests contain Read and Write requests.

Access Control also occurs in an SNMP entity when an SNMP notification message is generated by a Notification Originator Application.

The VACM defines a set of services that an application (such as a Command Responder or a Notification Originator Application) uses for checking access rights.

## Elements of VACM Model

RFC 3415 defines five elements that make up the VACM.

- [VACM Groups](#)
- [VACM Security Level](#)
- [VACM Contexts](#)

- [VACM MIB Views](#)
- [VACM Access Rights](#)

## VACM Groups

A particular combination of security model and security name can be mapped to a group. In other words, a group is a set of zero or more <security model, security name> tuples on whose behalf SNMP-managed objects can be accessed. Any given combination of security name and security model belongs to at most one group. A group is identified by a unique group name.

The VACM defines `vacmSecurityToGroupTable` (configurable by `Snmp_Cfg_Sec2Groups` in `snmp_cfg.c`) that lists the security model and security name combination mappings to a group name.

## VACM Security Level

The access rights for a group may vary by the security level of the message that contains the request.

For example, Nucleus SNMP can be configured to allow a read-only access to a particular group of SNMP-managed objects for any message communicated with no authentication and to allow write access for authenticated messages. Further, for certain sensitive MIB objects, Nucleus SNMP can be configured to allow access via request using a privacy service.

SNMPv1 and SNMPv2c only support unauthenticated security level, while SNMPv3 supports all three security levels that are unauthenticated, authenticated, and authenticated with privacy service.

The VACM requires that the security level be passed as input to the Access Control Subsystem when called to check access rights.

## VACM Contexts

A MIB context is a named subset of the object instances in the local MIB. Contexts provide a useful way of aggregating objects into collections with different access rights.

An object or object instance may appear in more than one context. A context is uniquely identified by a context name.

The VACM defines `vacmContextTable` (configurable by `Snmp_Cfg_Context_Names` in `snmp_cfg.c`) that lists the locally available contexts by context names.

## VACM MIB Views

For security reasons, VACM restricts the access rights of groups to only a subset of the SNMP-managed objects. To provide this capability, access to a context is via a “MIB View” which details a specific set of SNMP-managed objects within that context.

The MIB view is defined in terms of a collection or a family of subtrees, with each subtree being included or excluded from the view.

A subtree is simply a node in the MIB’s naming hierarchy plus all its subordinate elements. More formally, a subtree may be defined as the set of all objects and object instances that have a common ASN.1 OBJECT IDENTIFIER prefix to their OIDs. The longest common prefix of all the instances in the subtree is the object identifier of the parent of that subtree.

Associated with each access entry in `vacmAccessTable` (configurable by `Snmp_Cfg_Access` in `snmp_cfg.c`) are three MIB views, one each for read, write, and notify access.

Each MIB view consists of a set of view subtrees. Each MIB view either includes or excludes all the SNMP-managed objects and instances included in that subtree. In addition, a view mask is defined in order to reduce the amount of configuration information when fine-grained access control is required (for example, access control at the object instance level).

## VACM Access Rights

The View-based Access Control Model determines the access rights of a group. For a particular context to which a group has access using a particular security model and security level, that group’s access rights are given by a read-view, a write-view and a notify-view.

The read-view represents the set of object instances authorized for the group when reading objects. Reading objects occurs when processing a retrieval operation (when handling Read Class PDUs).

The write-view represents the set of object instances authorized for the group when writing objects. Writing objects occurs when processing a write operation (when handling Write Class PDUs).

The notify-view represents the set of object instances authorized for the group when sending objects in a notification.

## Inputs to Access Control Subsystem

Table 9-54 shows the inputs to the VACM when called for determining access rights.

**Table 9-54. Inputs to Access Control Subsystem**

Input	Description
Security Model	Security Model in use.

**Table 9-54. Inputs to Access Control Subsystem (cont.)**

Input	Description
Security Name	Security name has similar logical significance as user name. It identifies who is trying to access the object.
Security Level	Level of security. Its value can be unauthenticated, authenticated and authenticated with privacy.
View Type or Access Type	Type of operation under processing. Read, write or notify.
Context Name	Context containing SNMP-managed object.
Variable Name	OBJECT IDENTIFIER for SNMP-managed object or object instance on which access is required to be checked.

## Outputs of Access Control Subsystem

Table 9-55 shows the outputs of the VACM when called for checking access rights.

**Table 9-55. Outputs of Access Control Subsystem**

Output	Description
Access Allowed	An MIB view for the principal identified by this security name, with the request security model, security level, context name, and view type. It reports if the requested variable name is found and access is granted.
Not In View	The MIB view was found, but access is denied because the variable name is not included in the MIB view for the specified view type or access type.
No Such View	No MIB view for the specified view type or access type was found.
No Such Context	The context name is not supported. More formally, the context name is not found in vacmContextTable.
No Group Name	The group identified by the combination of security model and security name is not supported. That is, there is no such entry in security to group that maps a combination of security name and security model to a group name.
No Access Entry	There are no MIB views for this combination of security model, security name, security level, and context name. It reports if no such entry is found in the VACM Access table.

## Access Control Subsystem Processing

During processing of the VACM, the following steps are performed:

- The vacmContextTable is consulted for information about the SNMP context identified by the contextName. If information about this SNMP context is absent from the table, and an error indication (noSuchContext) is returned to the calling module.
- The vacmSecurityToGroupTable is consulted for mapping the security Model and securityName to a groupName. If the information about this combination is absent from the table, an error indication (noGroupName) is returned to the calling module.
- The vacmAccessTable is consulted for information about the groupName, context Name, securityModel and securityLevel. If information about this combination is absent from the table, an error indication (noAccessEntry) is returned to the calling module.

The following actions are taken in this step:

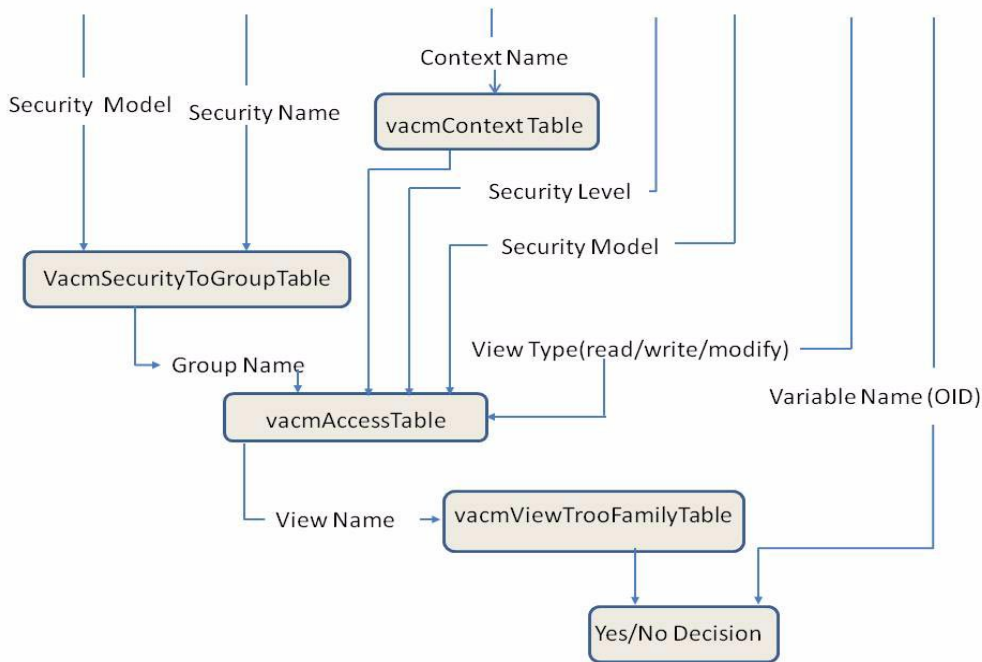
- If the viewType is “read”, then the read view is used for checking access rights.
- If the viewType is “write”, then the write view is used for checking access rights.
- If the viewType is “notify”, then the notify view is used for checking access rights.
- If the view to be used is the empty view (zero length viewName) then an error Indication (noSuchView) is returned to the calling module.
- The following actions are taken in this step:
  - If there is no view configured for the specified viewType, then an errorIndication (noSuchView) is returned to the calling module.
  - If the specified variableName (object instance) is not in the MIB view, then an errorIndication (notInView) is returned to the calling module.

Otherwise,

- The specified variableName is in the MIB view. Status information of success (accessAllowed) is returned to the calling module.

Figure 9-11 shows the processing of the VACM isAccessAllowed Process.

**Figure 9-11. VACM is AccessAllowed Process Flow**



When run for the first time (or when initialization fails), VACM needs to be configured. VACM can be configured either by defining the initial values in *include/networking/snmp\_cfg.h* and *networking/snmp/snmp\_cfg.c*, or through the serial interface at run-time.

Once configured, users can be added to VACM using the Nucleus SNMP manager. Refer to “[Nucleus SNMPv3 MIB](#)” on page 1669 for more information.

## include/networking/snmp\_cfg.h

### VACM\_CONTEXT\_TBL\_SIZE

This macro defines the number of context names that are added during configuration. This macro defines the size of the configurable array `Snm_Cfg_Context_Names` in *networking/snmp/snmp\_cfg.c*.

### VACM\_SEC2GRP\_TBL\_SIZE

This macro defines the number of Security to Group table entries that are added during configuration. This macro defines the number of entries in the configurable array `Snm_Cfg_Sec2Groups` in *networking/snmp/snmp\_cfg.c*.

## VACM\_ACCESS\_TBL\_SIZE

This macro defines the number of VACM Access table entries that are added during configuration. This macro defines the number of entries in the configurable array `Snmp_Cfg_Access` in *networking/snmp/snmp\_cfg.c*.

## VACM\_MIB\_VIEW\_TBL\_SIZE

This macro defines the number of views that are added during configuration. This macro defines the number of entries in the configurable array `Snmp_Cfg_Mib_View` in *networking/snmp/snmp\_cfg.c*.

## networking/snmp/snmp\_cfg.c

### Snmp\_Cfg\_Context\_Names

This global array holds the context names. To add a new context, add the entry containing the string defining the new context name.

### Snmp\_Cfg\_Sec2Groups

This global array holds the Security to Group entries to be configured during initialization. To add a new security to group entry, make the following entry in the array:

```
{Security Model, Security Name, Group Name}
```

`Snmp_Cfg_Sec2Groups` is an array of type [VACM\\_SEC2GROUP](#), defined in [SNMP Data Structures](#). An entry in this array defines a mapping of a combination of security model and security name to a group name.

### Snmp\_Cfg\_Access

This global array holds entries that define the read, write and notify access based upon security model, group name, context prefix, and security level. To add a new access entry, make the following entry in the array:

```
{Security Model, Group Name, Context Prefix, Security Level,  
Context Match, Read View, Write View, Notify View}
```

`Snmp_Cfg_Access` is an array of type [VACM\\_ACCESS](#), defined in [SNMP Data Structures](#). An entry in this table maps a combination of security model, group name, context name, and security level to MIB Views.

### Snmp\_Cfg\_Mib\_View

This global array holds entries that define which MIB objects are included in and are excluded from a MIB VIEW. To add a new MIB VIEW entry, make the following entry in the array:



```
{View Name, Subtree, Subtree Length, Family Mask, Mask Length,
Family Type}
```

Snm\_Cfg\_Mib\_View is an array of type [VACM\\_VIEWTREE](#), defined in [SNMP Data Structures](#).

## Notification Originator Application

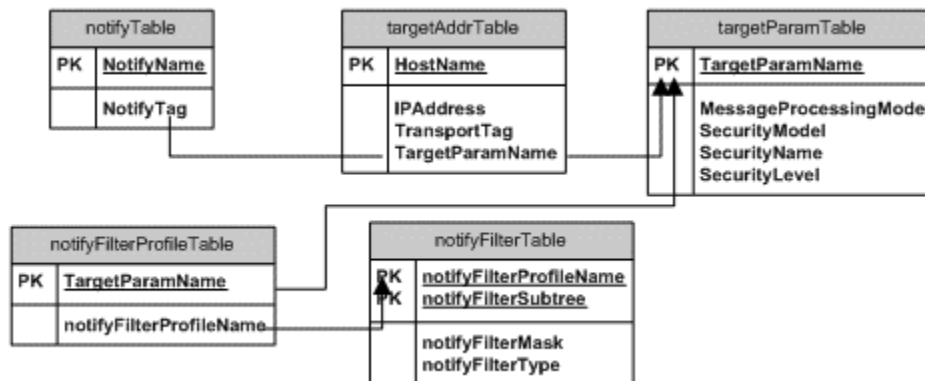
RFC 3413 completely defines the Notification Originator Application. This application is responsible for generating notifications.

The Notification Originator Application is configured by a call to `SNMP_Notification_Config` (*snmp\_no.c*). This routine is called internally by Nucleus SNMP during initialization. You are not required to call this routine. The following sections provide values that are assigned to each table. Refer to RFC 3413 for details on the following tables.

The Notification Originator Application can send any type of notification for example, linkUp, linkDown, coldStart, warmStart, authenticationFailure and any other notification you may want to send. The API routines for sending notifications or traps are [SNMP\\_Get\\_Notification\\_Ptr](#) and [SNMP\\_Notification\\_Ready](#).

[Figure 9-12](#) shows the relationship among the different tables used when sending out notifications or traps.

**Figure 9-12. Notification Originator Table Relationships**



## Notify Table

This table is configured using `Snm_Cfg_Notify_Tbl` in *snmp\_cfg.c*. By default, one entry is added to this table. By modifying `NOTIFY_TBL_SIZE` in *snmp\_cfg.h*, the number of entries to be added can be altered. The Notification Originator Application sends notifications for each entry in the Notify table to the IP Address found in corresponding entries in the Target Address table. The corresponding Target Address table entries are those that have the transport tag

specified in the Notify Table entry in their transport tag list. The Transport table entry refers to a Target Param table entry, which gives parameters used for encoding notifications.

The entry name or the Notify Name uniquely identifies an entry in the Notify table.

```
SNMP_NOTIFY_TABLE_CONFIG Snmp_Cfg_Notify_Tbl[NOTIFY_TBL_SIZE] =
{
    /* Entry name, Target Tag, Tag Length. */
    {"group1", "group1", 6}
};
```

## Target Address Table

This table is configured using `Snmp_Cfg_Tgr_Addr_Tbl` in *snmp\_cfg.c*. By default, five hosts are added to this table. By modifying `TGR_ADDR_TBL_SIZE` in *snmp\_cfg.h*, the number of hosts to be added can be altered. This table is used to send notifications to SNMP Managers. The parameters to be used for the notifications are defined by the Target Params table. Notification Originator Application only sends notification to the host defined in the Target Address table. This table is also used by the CBSM. Refer to the [“Advanced SNMP Topics”](#) on page 1693 for information on CBSM.

The following example shows configuration of `Snmp_Cfg_Tgr_Addr_Tbl`.

```
SNMP_TARGET_ADDR_TABLE Snmp_Cfg_Tgr_Addr_Tbl[TGR_ADDR_TBL_SIZE] = {
    /* Target name, Transport Protocol, IP Address,
     * IP Family (NU_FAMILY_IP for IPv4 and NU_FAMILY_IP6 for
     * IPv6, Unused variable (reserved for future use),
     * Tag List, Tag list length, Name of parameters entry to
     * use for this target.
     */
    {"Host1", SNMP_UDP, {192,200,100,50}, NU_FAMILY_IP, 0,
     "group1", 6, "NoAuthNoPriv-v1"},

    {"Host2", SNMP_UDP, {192,200,100,50}, NU_FAMILY_IP, 0,
     "group1", 6, "NoAuthNoPriv-v2"},

    {"Host3", SNMP_UDP, {192,200,100,50}, NU_FAMILY_IP, 0,
     "group1", 6, "NoAuthNoPriv-v3"},

    {"Host4", SNMP_UDP, {192,200,100,50}, NU_FAMILY_IP, 0,
     "group1", 6, "AuthNoPriv-v3"},

    {"Host5", SNMP_UDP, {192,200,100,50}, NU_FAMILY_IP, 0,
     "group1", 6, "AuthPriv-v3"}
};
```

## Target Params Table

This table is configured using `Snmp_Cfg_Tgr_Params_Tbl` in *snmp\_cfg.c*. By default, five entries are added to this table. By modifying `TGR_PARAMS_TBL_SIZE` in *snmp\_cfg.h*, the

number of entries to be added can be modified. This table is used to configure the parameters for sending notifications to SNMP Managers.

The following example shows the configuration of the `Snmpp_Cfg_Tgr_Params_Tbl`.

```
SNMP_TARGET_PARAMS_TABLE_CONFIG
Snmpp_Cfg_Tgr_Params_Tbl[TGR_PARAMS_TBL_SIZE] =
{
    /* Entry name, MP Model, Security Model, Security Name (or
     * Community Name for SNMPv1 and SNMPv2c), Security Level.
     */
    {"NoAuthNoPriv-v1", SNMP_VERSION_V1, SNMP_CBSM_V1, "public",
     SNMP_SECURITY_NOAUTHNOPRIV},
    {"NoAuthNoPriv-v2", SNMP_VERSION_V2, SNMP_CBSM_V2, "public",
     SNMP_SECURITY_NOAUTHNOPRIV},
    {"NoAuthNoPriv-v3", SNMP_VERSION_V3, SNMP_USM, "initial",
     SNMP_SECURITY_NOAUTHNOPRIV},
    {"AuthNoPriv-v3", SNMP_VERSION_V3, SNMP_USM, "initial",
     SNMP_SECURITY_AUTHNOPRIV},
    {"AuthPriv-v3", SNMP_VERSION_V3, SNMP_USM, "initial",
     SNMP_SECURITY_AUTHPRIV}
};
```

## Notify Filter Profile Table

This table is configured using `Snmpp_Cfg_Fltr_Prof_Tbl` in *snmpp\_cfg.c*. By default, five entries are added to this table. By modifying `FLTR_PROF_TBL_SIZE` in *snmpp\_cfg.h*, the number of entries to be added can be modified. This table is used to map a filter entry to the parameters entry.

The following example shows the configuration of the `Snmpp_Cfg_Fltr_Prof_Tbl`.

```
SNMP_NOTIFY_FILTER_PROFILE_TABLE_CONFIG
Snmpp_Cfg_Fltr_Prof_Tbl[FLTR_PROF_TBL_SIZE] =
{
    /* Params Name, Filter Name */
    {"NoAuthNoPriv-v1", "filter-1"},
    {"NoAuthNoPriv-v2", "filter-1"},
    {"NoAuthNoPriv-v3", "filter-1"},
    {"AuthNoPriv-v3", "filter-1"},
    {"AuthPriv-v3", "filter-1"}
};
```

## Notify Filter Table

This table is configured using `Snmpp_Cfg_Fltr_Tbl` in *snmpp\_cfg.c*. By default, two entries are added to this table. By modifying `FLTR_TBL_SIZE` in *snmpp\_cfg.h*, the number of entries to be added can be modified. This table defines the filters that are used before sending a notification.

The following example shows the configuration of the `Snmpp_Cfg_Fltr_Tbl`.

```
SNMP_NOTIFY_FILTER_TABLE_CONFIG Smmpp_Cfg_Fltr_Tbl[FLTR_TBL_SIZE] =
```

```
{
    /* Filter Name, Subtree, Subtree len, Filter Mask, Mask
     * Length (in octets), Filter Type.
     */
    {"filter-1", {1,3,6,1}, 4, {0xF0}, 1, 1}
};
```

## Optimizing Buffer Sizes

On certain enterprise MIBs, the GET\_BULK or GET\_NEXT requests have multiple variable bindings that may timeout or return a TOO\_BIG error. This requires updates in the buffer sizes for SNMPv1, SNMPv2, and SNMPv3 requests. SNMP\_V1\_BUFFERSIZE, SNMP\_V2\_BUFFERSIZE, SNMP\_V3\_BUFFERSIZE, and SNMP\_BUFSIZE may be insufficient for certain MIBs.

### Get Bulk

The Get Bulk request times out for SNMPv2 and SNMPv3 and the agent responds with a TOOBIG error status. In order to clarify, consider the following:

```
mibObject1  Size = 128
mibObject2  Size = 128
mibObject3  Size = 128
mibObject4  Size = 128
mibObject5  Size = 128
mibObject6  Size = 128
mibObject7  Size = 128
mibObject8  Size = 128
mibObject9  Size = 128
mibObject10 Size = 128
mibObject11 Size = 128
mibObject12 Size = 128
mibObject13 Size = 128
mibObject14 Size = 128
mibObject15 Size = 128
mibObject16 Size = 128
```

There is a Get Bulk Request on mibObject1 with Max Repetitions equal to 16. The total size of all objects shown equals 2048. These objects are to be encoded in the response PDU. The default value specified for SNMP\_BUFSIZE is 1024. The value of this macro must be greater than 2048 and a constant overhead equal to 60.

### Get Next

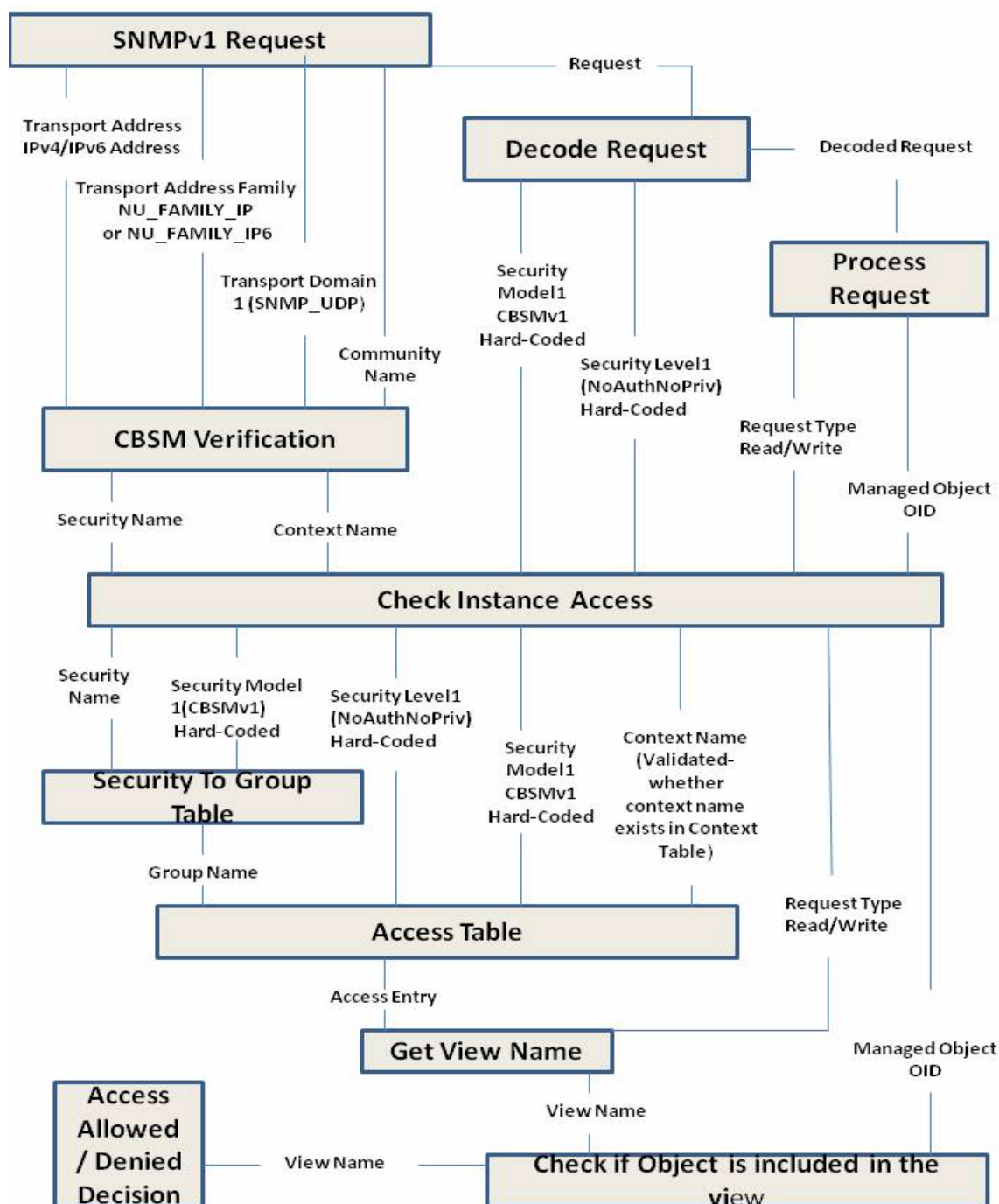
Get Next Request with a multiple variable bindings list equal to 16 may also time out in some enterprise MIBs. In this case, the agent does not respond to the request. The issue lies with decoding the packet in the given buffer size. It can be solved by modifying SNMP\_V1\_BUFFERSIZE, SNMP\_V2\_BUFFERSIZE, and SNMP\_V3\_BUFFERSIZE. Once again, consider the example. There are 16 objects in a multiple variable bindings list and a Get

Next Request is issued by the object Manager. Decoding these sixteen values exceeds the buffer size specified by the macros. Changing these values to a value greater than 2048 and a constant overhead equal to 60 in the example will resolve the issue.

## SNMPv1 Security Configurations

SNMPv1 uses the Community Based Security Model version 1(CBSMv1) and can allow the different access levels for the different community names in SNMPv1. However, there is no mechanism for having different access rights for the same community. [Figure 9-13](#) provides a better understanding of the access verification process for SNMPv1, starting from bottom to top.

**Figure 9-13. Access Verification for SNMPv1**



## VACM MIB View Configuration

Based on [Figure 9-13](#), it is quite obvious that the access to Managed Objects is granted to zero or more views. A managed object that is not included in any view is not accessible by any means. The view name is determined by the Request Type (read/write/notify) from the VACM Access table entry. By configuring MIB Views, you can define a set of Managed Objects that are accessible through the MIB View. You must have the appropriate VACM Access table entry to use the MIB View.

MIB Views can only be used by the entries in the VACM Access table as read/write/notify views. So, by having appropriate VACM Access table entries, access rights can be managed.

A VACM Access table entry is uniquely identified by:

- Group Name
- Context Prefix
- Security Model
- Security Level.

For SNMPv1, the value of Security Model and Security Level would always be 1 (SNMP\_CBSM\_V1) and 1 (SNMP\_SECURITY\_NOAUTHNOPRIV) respectively.

For SNMPv1, you can only alter Group Name and Context Prefix.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on the VACM Access table and VACM MIB Views. Two examples for designing MIB Views are provided below, and [How to Use VACM Views - VACM Access Table](#) provides instructions and examples for how to use VACM views.

In the following example, a MIB View named “OnlyEnterprisesAccess” is created that has access only to the enterprises MIB group subtree. Through this MIB View you can access the Managed Objects with OIDs like 1.3.6.1.4.1.x.x.x.x ... .

```
/* VACM MIB View that would only have access to enterprises MIB.
 */
VACM_VIEWTREE_STRUCT vacm_mib_view;

/* Accessible Subtree view. The value of 1.3.6.1.4.1 would also
 * includes 1.3.6.1.4.1.x.x.x.x. ...
 */
UINT32 accessible_subtree[] = {1,3,6,1,4,1};

/* Length of Subtree. */
UINT32 accessible_subtree_len = 6;

/* Family mask must be 6 bits long and all set. If
 * you don't want to use wild cards you will set all 6 bits to 1.
 * Consider a case, if you require access to all Managed
```

```

    * Objects in subtree with an OID like 1.3.6.1.x.1,
    * you need the family mask with value of 111101 binary hex {0xF4}.
    */

UINT8 subtree_family_mask[] = {0xFC};

/* Mask length in bytes would be 1, since six bits can't be
 * represented in fewer bytes.
 */
UINT32 subtree_family_mask_len = 1;

/* View name. */
CHAR view_name[] = "OnlyEnterprisesAccess";

/* Status to return success or error code */
STATUS status;

/* Clear out the structure. */
memset (&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the value of View Name. */
strcpy ((CHAR *) vacm_mib_view.vacm_view_name, view_name);

/* Set the value to Subtree. */
memcpy (vacm_mib_view.vacm_subtree, accessible_subtree,
        sizeof(accessible_subtree));

/* Set the Subtree length. */
vacm_mib_view.vacm_subtree_len = accessible_subtree_len;

/* Set the value of family mask. */
memcpy (vacm_mib_view.vacm_family_mask, subtree_family_mask,
        sizeof(subtree_family_mask));

/* Set the family mask length value. */
vacm_mib_view.vacm_mask_length = subtree_family_mask_len;

/* Allow access to the subtree. */
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Setting storage type. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set row status to 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add an MIB View Entry. */
status = VACM_InsertMibView(&vacm_mib_view);

/* Return success, or an error code. */
return (status);
```

The view created in the previous example can also be created by adding the following entry in `Snmp_Cfg_Mib_View` by editing the `networking/snmp/snmp_cfg.c` file and updating the value of `VACM_MIB_VIEW_TBL_SIZE`.



```
{"OnlyEnterprisesAccess", {1, 3, 6, 1, 4, 1}, 6, {0xFC}, 1,  
VACM_FAMILY_INCLUDED}
```

Nucleus SNMP enters into configuration mode when initialization from a file fails. A real file system is required for successful initialization from a file. During initialization from a file, Nucleus SNMP tries to restore its most recent state. When initialization from a file fails, Nucleus SNMP configures itself with the state defined in the configuration files.

In the next example, a MIB View named “mib2WithOutIfTable” is created that has access to the whole MIB-2 subtree, but not to the ifTable.

The following steps are included in the code example:

- Include the MIB-2 subtree in the MIB View.
- Exclude the ifTable subgroup from the MIB View.

```
/* View name */  
CHAR view_name[] = "mib2WithOutIfTable";  
  
/* MIB2-OID: OID of the subtree that is accessible to the MIB view */  
UINT32 mib2_oid_accessible[] = {1,3,6,1,2,1};  
  
/* Length of MIB2-OID */  
UINT32 mib2_oid_accessible_len = 6;  
  
/* The family mask must be 6 bits and all set. Since  
 * you don't want to use wild cards, all 6 bits are set to 1.  
 */  
UINT8 mib2_family_mask[] = {0xFC};  
  
/* Mask length in bytes is 1, since 6 bits can't be  
 * represented in fewer bytes.  
 */  
UINT32 mib2_family_mask_len = 1;  
  
/* ifTable OID: OID of the Subtree that must be  
 * excluded from the MIB View. For example: if you don't want  
 * to grant access to an ifTable through the MIB View you are  
 * creating.  
 */  
UINT32 ifTable_oid_excluded[] = {1,3,6,1,2,1,2,2};  
  
/* Length of ifTable OID */  
UINT32 ifTable_oid_len = 8;  
  
/* Family mask must be 8 bits long and all set. Since  
 * you don't want to use wild cards, set all 8 bits to 1.  
 */  
UINT8 ifTable_family_mask[] = {0xFF};  
  
/* Mask length in bytes would be 1, since 8 bits can't be  
 * represented in fewer bytes.  
 */  
UINT32 ifTable_family_mask_len = 1;
```

```
/* MIB View entry */
VACM_VIEWTREE_STRUCT vacm_mib_view;

/* Status to return success or an error code */
STATUS status;

/*****
/* The First step would grant access to the MIB-2 Subtree */
*****/

/* Clear out the MIB View entry. */
memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the value of View Name. */
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), view_name);

/* Set the value of Subtree. */
memcpy(vacm_mib_view.vacm_subtree, mib2_oid_accessible,
        sizeof(mib2_oid_accessible));

/* Set the value of subtree length. */
vacm_mib_view.vacm_subtree_len = mib2_oid_accessible_len;

/* Set the value to family mask. */
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));

/* Set the value of family mask length. */
vacm_mib_view.vacm_mask_length = mib2_family_mask_len;

/* Set the value of family type to 'included' to represent that
 * the mib-2 subtree is accessible from this MIB View.
 */
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the storage type to read-only. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the row status as 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the MIB View Entry. */
status = VACM_InsertMibView(&vacm_mib_view);

/* If the MIB View Entry was not added successfully, then return
 * an error code.
 */
if (status != NU_SUCCESS)
{
    /* Return an error code. */
    return (status);
}

/*****
/* The second step is to exclude ifTable from this MIB View. */
*****/
```

```
/* **** */

/* Clear out the MIB View entry structure. */
memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));

/* Set the View Name. */
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), view_name);

/* Set the value of subtree to ifTable-OID. */
memcpy(vacm_mib_view.vacm_subtree, ifTable_oid_excluded,
       sizeof(ifTable_oid_excluded));

/* Set the length of OID. */
vacm_mib_view.vacm_subtree_len = ifTable_oid_len;

/* Set the value of Family Mask. */
memcpy(vacm_mib_view.vacm_family_mask, ifTable_family_mask,
       sizeof(ifTable_family_mask));

/* Set the length of Family Mask. */
vacm_mib_view.vacm_mask_length = ifTable_family_mask_len;

/* You don't want to grant access to this MIB View. */
vacm_mib_view.vacm_family_type = VACM_FAMILY_EXCLUDED;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the storage type to read-only. */
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status to 'active'. */
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif

status = VACM_InsertMibView(&vacm_mib_view);

/* Return success, or an error code. */
return (status);
```

The view created in the previous example can also be created by adding the following two entries in the `Snmplib_Mib_View` by editing the `networking/snmp/snmp_cfg.c` file and updating the value of `VACM_MIB_VIEW_TBL_SIZE` in the `include/networking/snmp_cfg.h` file.

```
{ "mib2WithoutIfTable", {1, 3, 6, 1, 2, 1}, 6, {0xFC}, 1,
  VACM_FAMILY_INCLUDED},
{ "mib2WithoutIfTable", {1, 3, 6, 1, 2, 1, 2, 2}, 8, {0xFF}, 1,
  VACM_FAMILY_EXCLUDED}
```

Nucleus SNMP enters into configuration mode when initialization from a file fails. A real file system is required for successful initialization from file. During initialization from a file, Nucleus SNMP tries to restore its most recent state. When initialization from a file fails Nucleus SNMP configures itself with the state defined in the configuration files.

An entry in the MIB View table is uniquely identified by:

- View Name
- Subtree OID
- Subtree length

Multiple entries in the MIB View table can have same the view names with different Subtree OIDs (Subtree OID length is omitted because it is nothing, but it is there to complete what is defined by Subtree OID).

Nucleus SNMP, while validating access on a particular managed object, uses the MIB View entry with close prefix OID match. For example, `ifIndex` (1.3.6.1.2.1.2.2.1.1) falls in both subtree MIB2 (1.3.6.1.2.1) and `ifTable` (1.3.6.1.2.1.2.2), Nucleus SNMP selects the `ifTable` because it has eight prefix OID matching indices lesser than prefix matching indices with the MIB-2 subtree that is six. After selecting the appropriate entry, Nucleus SNMP checks to determine if this subtree is set to be excluded or included (to allow or disallow access) in the MIB View based upon the value of family type. However, if no entry exists with the prefix OID match for a given view name, then Nucleus SNMP simply disallows the access.

## How to Use VACM Views - VACM Access Table

This section walks you through how to create three VACM Access table entries for the group names “`mib2ReadWriteNotify`”, “`mib2ReadWrite`” and “`mib2ReadOnly`”, with access rights according to their names. An example is provided for each of the steps.

### Procedure

1. The first step is to create the MIB View named “`mib2View`” that has access to the MIB-2 subtree.

```
UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT vacm_mib_view;
STATUS                status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#ifdef INCLUDE_MIB_VACM == NU_TRUE
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);
```

2. Create a VACM Access table entry for the group name “`mib2ReadWrite Notify`” that has read/write/notify access to the MIB View “`mib2View`” through SNMPv1.

```
VACM_ACCESS_STRUCT          vacm_access_entry;
STATUS                      status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadWriteNotify");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. However, when you enable
 * prefix matching by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in access entry ("ab",
 * for example) would also match a context name like "abxxxx", as
 * well as the intended "ab". An entry is required in the
 * Context table for the context name used in the Community table
 * entry, otherwise access will not be granted. However, the
 * context prefix in in access entry is not require to be
 * registered in the Context table. The API for registering
 * a Context Name is VACM_InsertContext. An entry in the context
 * table can also be added in configuration mode by modifying
 * the Snmp_Cfg_Context_Names ("networking/snmp/snmp_cfg.c") and
 * VACM_CONTEXT_TBL_SIZE ("include/networking/snmp_cfg.h").
 */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of the security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* The Security Level in SNMPv1 is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make the MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

3. The third step is to create the VACM Access table entry for the group name “mib2ReadWrite” with the access rights of read-write on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT          vacm_access_entry;
STATUS                      status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* The Security Level in SNMPv1 is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name. This results in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

4. The fourth step is to create the VACM Access table entry for the group name “mib2ReadOnly” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT          vacm_access_entry;
STATUS                      status;
```

```
/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* The Security Level in SNMPv1 is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Clear out write view. This will result in no Write Access.
 * Notify view is already being cleared out, so there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name. This results in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

## Results

You have now created three VACM Access table entries for the group names “mib2ReadWriteNotify”, “mib2ReadWrite” and “mib2ReadOnly”, with access rights according to their names. The previous configuration can also be created by editing *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* as follows:

Add the following entry to *Snm\_Cfg\_Mib\_View* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_MIB\_VIEW\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```
{ "mib2View", {1, 3, 6, 1, 2, 1}, 6, {0xFC}, 1,  
  VACM_FAMILY_INCLUDED }
```

Add the following entry in `Snm_Cfg_Access` in `networking/snmp/snmp_cfg.c` and update the value of `VACM_ACCESS_TBL_SIZE` in `include/networking/snmp_cfg.h` accordingly.

```
{SNMP_CBSM_V1, "mib2ReadWriteNotify", "", VACM_CTXTMATCH_EXACT,  
  SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", "mib2View"},  
{SNMP_CBSM_V1, "mib2ReadWrite", "", VACM_CTXTMATCH_EXACT,  
  SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", ""},  
{SNMP_CBSM_V1, "mib2ReadOnly", "", VACM_CTXTMATCH_EXACT,  
  SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "", ""}
```

Nucleus SNMP enters into the configuration mode when initialization from files fails. A real file system is required for successful initialization from file. During initialization from files, Nucleus SNMP tries to restore its most recent state. When initialization from file fails, Nucleus SNMP configures itself with the state defined in the configuration files.

## Related Topics

[Advanced SNMP Topics](#)

[VACM MIB View Configuration](#)

## Access Entries - The Security to Group Table

Access entries are only accessible by the group names assigned to them. For accessing group names, an entry must be added to the [VACM Security To Group](#) table.

An entry in the Security to Group table maps a combination of the security model and the security name to a group name. A group is a set of tuples of security model and security names. Different combinations of security models and security names can be mapped to groups; however, each combination of security model and security name cannot be mapped to more than one group. The groups are designed to grant the security model independent access rights, given that each different combination of security model and security name can be mapped to single group.

For SNMPv1, the security model is always 1 (`SNMP_CBSM_V1`); however, you can change the security name. Given that the security model is fixed, SNMPv1 allows you to grant different access rights using different security names.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on the VACM Security to Group table.



## Mapping Community Names into the Community Table

Nucleus SNMP derives the security name from the community name by accessing an entry from the Community table. This results in the different access rights for the different communities. However, using the same security name for different communities results in the same access rights for the different communities.

There are two different fields for the community index and the community name. An entry in the Community table is uniquely identified by community index. Nucleus SNMP uses the same values for both the community index and community name fields; however, advanced user may use different values.

To complete the task of setting up the customized SNMPv1 securities, entries must be added to the Community table and the Security to Group table. The following procedure walks you through how to create three communities named “mib2RdWtNotComm”, “mib2RdWtComm” and “mib2RdOnlyComm”, with the access rights of read-write-notify, read-write, and read-only on the MIB-2 Subtrees respectively.

### Procedure

1. The first step is to create an MIB View named “mib2View” that has access to the MIB-2 subtree.

```
UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT vacm_mib_view;
STATUS                status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);
```

2. The second step is to create the VACM Access table entry for the group name “mib2Read WriteNotify” that has read/write/notify access to the MIB-2 Subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT    vacm_access_entry;
STATUS                status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
```

```
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
        "mib2ReadWriteNotify");

/* Set the value of the context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled; however, when you enable
 * prefix matching by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in the access entry of "ab"
 * also matches context names like "abxxxx". An entry must
 * exist in the Context table for each context name used in the
 * Community table entry, otherwise access will not be granted.
 * But, it is not a requirement that the context prefix in the
 * access entry be registered in the Context table. The API
 * that registers a context Name is VACM_InsertContext. An entry in
 * the context table can also be added in the configuration mode by
 * modifying the Snmp_Cfg_Context_Names in the
 * networking/snmp/snmp_cfg.c file and the VACM_CONTEXT_TBL_SIZE
 * in the include/networking/snmp_cfg.h file.
 */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Security Level in SNMPv1 to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make the MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

3. The third step is to create the VACM Access table entry for the group name “mib2Read Write” with the access rights of read-write on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT    vacm_access_entry;
STATUS                status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));
```

```
/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Security Level in SNMPv1 is set to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level = SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name. This results in no notify access. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

4. The fourth step is to create the VACM Access table entry for the group name “mib2Read Only” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;
```

```
/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Security Level in SNMPv1 is set to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Clear out the write view. This will result in no write Access.
 * The Notify view is already being cleared out, so there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name to remove notify access. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

You now have added the VACM Access table entries for the groups named “mib2ReadWrite Notify”, “mib2ReadWrite”, and “mib2ReadOnly” with the access rights according to their names. Now you must add the relevant entries in the Security to Group table for the SNMP\_CBSM\_V1 security model and the security names of “mib2RdWtNotSec”, “mib2RdWtSec” and “mib2RdOnlySec” respectively.

5. The fifth step is to map the combination of the security model, CBSMv1, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.

```
VACM_SEC2GROUP_STRUCT    vacm_sec2group_entry;
STATUS                   status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
    "mib2RdWtNotSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
    "mib2ReadWriteNotify");
```

```
#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

6. The sixth step is to map the combination of the security model, CBSMv1, and the security name “mib2RdWtSec” to the group named “mib2ReadWrite”.

```
VACM_SEC2GROUP_STRUCT    vacm_sec2group_entry;
STATUS                   status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWrite");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

7. The seventh step is to map the combination of the security model, CBSMv1, and the security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.

```
VACM_SEC2GROUP_STRUCT    vacm_sec2group_entry;
STATUS                   status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdOnlySec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;
```

```
/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadOnly");

#ifdef (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);

return (status);
```

You now have three security names (“mib2RdWtNotSec”, “mib2RdWtSec”, and “mib2RdOnlySec”) with access rights of read-write-notify, read-write and read-only on the MIB-2 subtree respectively. To use these access rights, you must add entries in the Community table that use these security names.

8. The eighth step is to create the Community table entry with the community name “mib2RdWtNotComm” that uses the security name “mib2RdWtNotSec”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index,
       "mib2RdWtNotComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtNotComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtNotComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtNotSec");

/* Get the Engine Id.  SNMP determines the Engine ID based upon
 * the local IP Address.  A complex procedure is involved in
 * generating this value; therefore, it's better to retrieve the
 * value of Engine ID from Nucleus SNMP.
 */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return error code. */
```

```
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#ifdef INCLUDE_MIB_CBSM == NU_TRUE
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Community table. */
status = CBSM_Add_Community(&community);
```

9. The ninth step is to create the Community table entry with the community name “mib2RdWt Comm” that is using the security name “mib2RdWtSec”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdWtComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return error code. */
```

```
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#ifdef INCLUDE_MIB_CBSM == NU_TRUE
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Community table. */
status = CBSM_Add_Community(&community);
```

10. The tenth step is to create the Community table entry with the community name “mib2Rd OnlyComm” that is using the security name “mib2RdOnlySec”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdOnlyComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
(UINT8)strlen("mib2RdOnlyComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdOnlyComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdOnlySec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return an error code. */
```



```

if (status != NU_SUCCESS)
    return (status);

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Community table. */
status = CBSM_Add_Community(&community);

```

The previous configuration can also be created by editing *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* as follows:

Add the following entry to *Snm\_Cfg\_Mib\_View* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_MIB\_VIEW\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```
{ "mib2View", {1, 3, 6, 1, 2, 1}, 6, {0xFC}, 1, VACM_FAMILY_INCLUDED }
```

Add the following entries in *Snm\_Cfg\_Access* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_ACCESS\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```

{SNMP_CBSM_V1, "mib2ReadWriteNotify", "", VACM_CTXTMATCH_EXACT,
 SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", "mib2View"},
{SNMP_CBSM_V1, "mib2ReadWrite", "", VACM_CTXTMATCH_EXACT,
 SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", ""},
{SNMP_CBSM_V1, "mib2ReadOnly", "", VACM_CTXTMATCH_EXACT,
 SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "", ""}

```

Add the following entries to *Snm\_Cfg\_Sec2Groups* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_SEC2GRP\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```

{SNMP_CBSM_V1, "mib2RdWtNotSec", "mib2ReadWriteNotify"}
{SNMP_CBSM_V1, "mib2RdWtSec", "mib2ReadWrite"}
{SNMP_CBSM_V1, "mib2RdOnlySec", "mib2ReadOnly"}

```

And add the following entries to `CBSM_Community_Table` in `networking/snmp/snmp_cfg.c` and update the value of `CBSM_MAX_COMMUNITIES` in `include/networking/snmp_cfg.h` accordingly.

```
{ "mib2RdWtNotComm", "mib2RdWtNotSec", "group1", "" }  
{ "mib2RdWtComm", "mib2RdWtSec", "group1", "" }  
{ "mib2RdOnlyComm", "mib2RdOnlySec", "group1", "" }
```

Nucleus SNMP enters into configuration mode when initialization from files fails. A real file system is required for successful initialization from file. During initialization from files, Nucleus SNMP tries to restore its most recent state. When initialization from file fails, Nucleus SNMP configures itself with the state defined in the configuration files.

## Results

There are now three communities named “mib2RdWtNotComm”, “mib2RdWtComm”, and “mib2RdOnlyComm”, with access rights of read-write-notify, read-write and read-only on MIB-2 respectively.

## Related Topics

[Advanced SNMP Topics](#)

## Adding Security Using the Target Address Table

In SNMPv1, you can set up different access rights for different communities. This may lead to concern that, if an un-authorized person learns the name of a community, the person may have unauthorized access to the Nucleus SNMP Agent. However, in addition to the previously discussed security checks, SNMPv1 has an additional check to validate the IP Address from where the SNMP request is received. The table used to perform this check is the Target Address table.

The value of the transport tag in the Community table entry decides which entry is used from the Target Address table. The entry in the Target Address is used to validate an IP address. An entry in the Target Address table can have either a single or multiple authorized IP addresses.

A request received from an un-authorized IP Address will not be processed by the Nucleus SNMP Agent. You can configure SNMPv1 securities at the IP Address level as well.

The following procedure and example assumes that the “mib2RdWtNotComm” and “mib2RdWtComm” communities can only be accessible from a particular IP Address (say 192.168.0.14). Additionally, you would like to grant read-only access to the community named “mib2RdOnlyComm” that is only accessible by a pool of IP addresses from 192.168.0.0 to 192.168.0.255. This example is an extension of the example for [Mapping Community Names into the Community Table](#).

## Procedure

1. The first step is to create a MIB View named “mib2View” that has access to the MIB-2 subtree.

```

UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT vacm_mib_view;
STATUS                status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);

```

2. The second step is to create the VACM Access table entry for the group name “mib2ReadWrite Notify” that has read/write/notify access to the MIB-2 subtree using the MIB View “mib2 View” created in the first step.

```

VACM_ACCESS_STRUCT    vacm_access_entry;
STATUS                status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *) vacm_access_entry.vacm_group_name,
        "mib2ReadWriteNotify");

/* Set the value of context prefix. */
strcpy((CHAR *) vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled; however, when you enable
 * prefix matching by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in the access entry of "ab"
 * also matches context names like "abxxxxx". An entry must
 * exist in the Context table for each context name used in the
 * Community table entry, otherwise access will not be granted.
 * But, it is not a requirement that the context prefix in the
 * access entry be registered in the Context table. The API
 * that registers a context Name is VACM_InsertContext. An entry in
 * the context table can also be added in the configuration mode by
 * modifying the Snmp_Cfg_Context_Names in the
 * networking/snmp/snmp_cfg.c file and the VACM_CONTEXT_TBL_SIZE
 * in the include/networking/snmp_cfg.h file.
 */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

```

```
/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Security Level in SNMPv1 is set to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make the MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

3. The third step is to create the VACM Access table entry for the group name “mib2Read Write” with the access rights of read-write on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out the access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
    "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Security Level in SNMPv1 is set to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");
```

```
/* Make the MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name, removing notify access. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

4. The fourth step is to create the VACM Access table entry for the group name “mib2ReadOnly” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching is enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V1;

/* Security Level in SNMPv1 is set to 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make the MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Clear out the write view, removing Write Access.
 * Notify view is already being cleared out. So, there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name, removing notify access. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
```

```
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Access Entry to the VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

You now have added the VACM Access table entries for the groups named “mib2ReadWriteNotify”, “mib2ReadWrite”, and “mib2ReadOnly” with access rights according to their names. The next step is to add the relevant entries in the Security to Group table with the security model set to SNMP\_CBSM\_V1 and security names of “mib2RdWtNotSec”, “mib2RdWtSec” and “mib2RdOnlySec” respectively.

5. The fifth step is to map the combination of the security model, CBSMv1, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                     status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtNotSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWriteNotify");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

6. The sixth step is to map the combination of the security model, CBSMv1, and the security name “mib2RdWtSec” to the group named “mib2ReadWrite”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                     status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));
```

```
/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWrite");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

7. The seventh step is to map the combination of the security model, CBSMv1, and the security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                     status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdOnlySec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V1;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadOnly");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);

return (status);
```

You now have three security names (“mib2RdWtNotSec”, “mib2RdWtSec”, and “mib2RdOnlySec”) with the access rights of read-write-notify, read-write, and read-only on

the MIB-2 subtree respectively. To use these access rights, you must add entries in the Community table. You can have different access rights for different communities.

Prior to the creating communities, you must create the Target Address table entries.

8. The eighth step is to create the Transport Address table entry named “HostRdWrNot” with the transport tag “RdWrNotTag” and the IP Address 192.168.0.14.

```
SNMP_TARGET_PARAMS_TABLE    target_param_entry;
SNMP_TARGET_ADDRESS_TABLE    target_entry;
STATUS                       status;

/* Target Address on which you grant read-write-notify
 * access.
 */
UINT8      read_write_target_addr[IP_ADDR_LEN] = {192,168,0,14};

/* Clear out the Target Address entry. */
memset(&target_entry, 0, sizeof(target_entry));

/* Set the name of Target table Entry. */
strcpy(target_entry.snmp_target_addr_name, "HostRdWrNot");

/* Set the Target Address. */
memcpy(target_entry.snmp_target_addr_tAddress,
        read_write_target_addr, IP_ADDR_LEN);

/* Set the Target Address Family. */
target_entry.snmp_target_addr_tfamily = NU_FAMILY_IP;

/* Disable Target Address masking so only a single
 * address can be accessible by this Target Address entry.
 */
target_entry.snmp_ext_enabled = NU_FALSE;

/* Set the transport tag that is used by Community table
 * entry.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_tag_list,
        "RdWrNotTag");

/* Set the length of transport tag. */
target_entry.tag_list_len = (UINT32)strlen("RdWrNotTag");

/* Set the name of the Target Params table entry. You will create
 * its entry in the Target Params table later in this example.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_params,
        "RdWrNotTag-v1");

/* Set the length of Target Params. */
target_entry.params_len = (UINT32)strlen("RdWrNotTag-v1");

/* Set the transport domain. */
target_entry.snmp_target_addr_tDomain = SNMP_UDP;

/* Set the time out value. */
```



```
target_entry.snmp_target_addr_time_out = 1000;

/* Set the retry count. */
target_entry.snmp_target_addr_retry_count = 3;
#if (INCLUDE_MIB_TARGET == NU_TRUE)
    /* Set the storage type. */
    target_entry.snmp_target_addr_storage_type =
        SNMP_STORAGE_READONLY;

    /* Set the Target Address entry as 'active'. */
    target_entry.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Target Address entry. */
status = SNMP_Add_Target(&target_entry);

/* If you failed to add the Target Address entry then return error
 * code.
 */
if (status != NU_SUCCESS)
    return status;

/* Clear out the Target Params table entry. */
memset(&target_param_entry, 0, sizeof(target_param_entry));

/* Set the Target Params table entry name. */
strcpy(target_param_entry.snmp_target_params_name,
        "RdWrNotTag-v1");

/* Set the Message Processing Model. */
target_param_entry.snmp_target_params_mp_model = SNMP_VERSION_V1;

/* Set the Security Name. */
strcpy(target_param_entry.snmp_target_params_security_name,
        "mib2RdWtNotSec");

/* Set the security model. */
target_param_entry.snmp_target_params_security_model =
        SNMP_CBSM_V1;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
    /* Set the Target Params entry's storage type. */
    target_param_entry.snmp_target_params_storage_type =
        SNMP_STORAGE_READONLY;

    /* Set the target param table entry as 'active'. */
    target_param_entry.snmp_target_params_row_status =
        SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Target Params table. */
status = SNMP_Add_Params(&target_param_entry);
```

9. The ninth step is to create the Transport Address table entry named “HostRdWrNot” with the transport tag “HostRdOnlyTag”, the IP Address of 192.168.0.14, and the IP Address Mask 255.255.255.0, so that all the IP Addresses from 192.168.0.0 to 192.168.0.255 are included.

```
SNMP_TARGET_PARAMS_TABLE    target_param_entry;
SNMP_TARGET_ADDRESS_TABLE    target_entry;
STATUS                       status;

/* Target Address on which you grant access read-only access. */
UINT8 read_only_target_addr[IP_ADDR_LEN] = {192,168,0,0};

/* You want to grant read-only access to a pool of IP addresses
 * from 192.168.0.0 to 192.168.0.255. Zero in the address mask
 * represents a wild card and 192.168.0.14 is included in
 * that range. Access rights for Target Address 192.168.0.14
 * are then determined by the community name it uses in its
 * requests.
 */
UINT8 read_only_target_mask[IP_ADDR_LEN] = {255, 255, 255, 0};

/* Clear out the Target Address entry. */
memset(&target_entry, 0, sizeof(target_entry));

/* Set the name of Target table entry. */
strcpy(target_entry.snmp_target_addr_name, "HostRdWrNot");

/* Set the Target Address. */
memcpy(target_entry.snmp_target_addr_tAddress,
        read_only_target_addr, IP_ADDR_LEN);

/* Set the Target Address Family. */
target_entry.snmp_target_addr_tfamily = NU_FAMILY_IP;

/* Enable Target Address masking so you can use a pool
 * of IP addresses with this Target Address entry.
 */
target_entry.snmp_ext_enabled = NU_TRUE;

/* Set the Target Address mask. */
memcpy(target_entry.snmp_target_addr_tmask,
        read_only_target_mask, IP_ADDR_LEN);

/* Set the transport tag it is used by Community table
 * entry.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_tag_list,
        "HostRdOnlyTag");

/* Set the length of transport tag. */
target_entry.tag_list_len = (UINT32)strlen("RdWrNotTag");

/* Set the name of Target Params table entry. You will create its
 * entry in the Target Params table later in this example.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_params,
        "ParamRdOnly-v1");

/* Set the length of Target Params. */
target_entry.params_len = (UINT32)strlen("RdWrNotTag-v1");

/* Set the transport domain. */
target_entry.snmp_target_addr_tDomain = SNMP_UDP;
```

```
/* Set the time out value. */
target_entry.snmp_target_addr_time_out = 1000;

/* Set the retry count. */
target_entry.snmp_target_addr_retry_count = 3;

/* Set the maximum message size. */
target_entry.snmp_target_addr_mms = 500;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
    /* Set the storage type. */
    target_entry.snmp_target_addr_storage_type =
        SNMP_STORAGE_READONLY;

    /* Set the Target Address entry as 'active'. */
    target_entry.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Target Address entry. */
status = SNMP_Add_Target(&target_entry);

/* If you fail to add the Target Address entry then an return error
 * code.
 */
if (status != NU_SUCCESS)
    return status;

/* Clear out Target Params table entry. */
memset(&target_param_entry, 0, sizeof(target_param_entry));

/* Set the Target Params table entry name. */
strcpy(target_param_entry.snmp_target_params_name,
        "ParamRdOnly-v1");

/* Set the Message Processing Model. */
target_param_entry.snmp_target_params_mp_model = SNMP_VERSION_V1;

/* Set the Security Name. */
strcpy(target_param_entry.snmp_target_params_security_name,
        "mib2RdOnlySec");

/* Set the security model. */
target_param_entry.snmp_target_params_security_model =
        SNMP_CBSM_V1;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
    /* Set the Target Params entry's storage type. */
    target_param_entry.snmp_target_params_storage_type =
        SNMP_STORAGE_READONLY;

    /* Set the target para table entry as 'active'. */
    target_param_entry.snmp_target_params_row_status =
        SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Target Params table. */
status = SNMP_Add_Params(&target_param_entry);
```

10. The tenth step is to create the Community table entry with the community name “mib2RdWtNotComm” that is using the security name “mib2RdWtNotSec” and the transport tag “RdWrNotTag”.

```
CBSM_COMMUNITY_STRUCT      community;
STATUS                     status;
UINT8                     engine_id[50];
UINT32                    engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index,
        "mib2RdWtNotComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtNotComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtNotComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtNotSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return an error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. Nucleus SNMP searches
 * all the entries with this transport tag. This really
 * helps in mapping multiple IP address pools to a
 * single community.
 */
strcpy((CHAR *)community.cbsm_transport_tag, "RdWrNotTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("RdWrNotTag");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
/* Set the storage type. */
community.cbsm_storage_type = SNMP_STORAGE_READONLY;
```

```
/* Set the status of community to active. */
community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Community table. */
status = CBSM_Add_Community(&community);
```

11. The eleventh step is to create the Community table entry with the community name “mib2RdWtComm” that is using the security name “mib2RdWtSec” and the transport tag “RdWrNotTag”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdWtComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len = (UINT8)strlen("mib2RdWtComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return an error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "RdWrNotTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("RdWrNotTag");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
/* Set the storage type. */
community.cbsm_storage_type = SNMP_STORAGE_READONLY;
```

```
        /* Set the status of community to active. */
        community.cbsm_status = SNMP_ROW_ACTIVE;
    #endif

    /* Add the entry to the Community table. */
    status = CBSM_Add_Community(&community);
```

12. The twelveth and the final step is to create the Community table entry with the community name “mib2RdOnlyComm” that is using the security name “mib2RdOnlySec” and the transport tag “HostRdOnlyTag”.

```
CBSM_COMMUNITY_STRUCT    community;
UINT8                    engine_id[50];
UINT32                   engine_id_len;
STATUS                   status;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdOnlyComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdOnlyComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdOnlyComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdOnlySec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get the Engine ID, return an error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "HostRdOnlyTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("HostRdOnlyTag");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;
```

```
/* Set the status of community to active. */
community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Add the entry to the Community table. */
status = CBSM_Add_Community(&community);
```

This configuration can not be created by setting the values in *networking/snmp/snmp\_cfg.c* or *include/networking/snmp\_cfg.h* because Nucleus SNMP does not provide an interface to create Transport IP Address pools in the *Snp\_Cfg\_Tgr\_Addr\_Tbl*.

## Results

The “mib2RdWtNotComm” and “mib2RdWtComm” communities are setup to only be accessible from a particular IP Address (say 192.168.0.14); read-only access has been granted to the community named “mib2RdOnlyComm” that is only accessible by a pool of IP addresses from 192.168.0.0 to 192.168.0.255.

## Related Topics

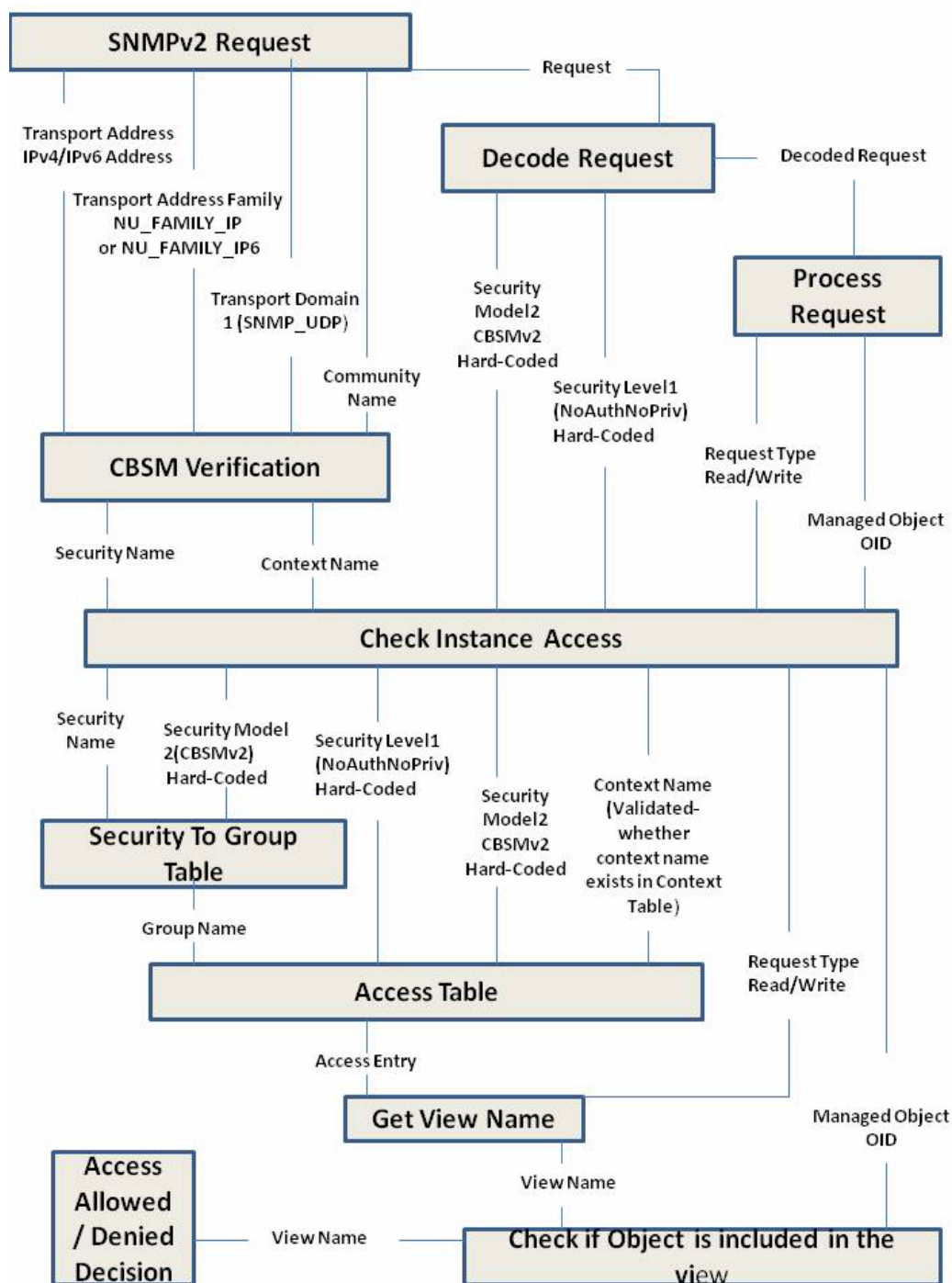
[Access Entries - The Security to Group Table](#)   [Mapping Community Names into the Community Table](#)

[Advanced SNMP Topics](#)

# SNMPv2c Security Configurations

You can allow different access levels for different community names in SNMPv2c. SNMPv2c Security Configurations are almost similar to security configuration of SNMPv1. This discussion here is also self-contained. You are not required to go through SNMPv1 Security Configuration. However there is no mechanism for having different access rights for same community. Figure A-2 provides a better understanding using a bottom up view of “Access Verification for SNMPv2c”.

**Figure 9-14. Access Verification for SNMPv2c**





## VACM MIB View

After having a look at the figure, it is quite obvious that access to Managed Objects is granted to zero or more views. A managed object that is not included in any view will not be accessible by any mean. The view name is determined by Request Type (read/write/notify) and Access Entry. So, discussion until now leads to conclusion that by configuring MIB Views, you make a set of Managed Objects that are accessible through a MIB View. And a requirement of an appropriate Access Entry to use that MIB Views is also there.

Examples for configuring VACM MIB Views can be found under “SNMPv1 Security Configurations”.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on VACM MIB Views.

## How to Use VACM Views - VACM Access Table

These MIB Views can only be used by the entries in the VACM Access table as read/write/notify views. So, by having appropriate VACM Access table entries, access rights can be managed.

A VACM Access table entry is uniquely identified by:

- Group Name
- Context Prefix
- Security Model
- Security Level.

For SNMPv2c, the value of Security Model and Security Level would always be 2 (SNMP\_CBSM\_V2) and 1 (SNMP\_SECURITY\_NOAUTHNOPRIV) respectively.

Keeping focus on SNMPv2c, you can only alter Group Name and Context Prefix.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on VACM Access table.

## Example

In the next example, you create three VACM Access table entries for the group names “mib2ReadWriteNotify”, “mib2ReadWrite” and “mib2ReadOnly”, with access rights according to their names.

The following are the steps involved in the code example:

- Create a MIB View named “mib2View” that have access to the MIB-2 subtree.

- Create a VACM Access table entry for the group name “mib2ReadWrite Notify” that has read/write/notify access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadWrite” that has read/write access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadOnly” that has read-only access to the MIB View “mib2View” through SNMPv2c.

The first step is to create the MIB View named “mib2View” that have access to the MIB-2 subtree.

```
UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT  vacm_mib_view;
STATUS                 status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);
```

The second step is to create the VACM Access table entry for the group name “mib2ReadWrite Notify” that has read/write/notify access to the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT     vacm_access_entry;
STATUS                 status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *) vacm_access_entry.vacm_group_name,
        "mib2ReadWriteNotify");

/* Set the value of context prefix. */
strcpy((CHAR *) vacm_access_entry.vacm_context_prefix, "");

/* However, when you enableExact context matching enabled.
 * by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in access entry say
 * "ab" would also match context name like "abxxxx" and of
 * course with "ab". Please note that, an entry is required to
 * be there in Context table for the context name used in
 * Community table entry, otherwise access will not be granted.
```

```
* But, context prefix in access entry is not require to
* registered in Context table. The API routine for registering
* a Context Name is VACM_InsertContext. An entry in context
* table can also be added in configuration mode by modifying
* Snmp_Cfg_Context_Names ("networking/snmp/snmp_cfg.c") and
* VACM_CONTEXT_TBL_SIZE ("include/networking/snmp_cfg.h").
*/
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The third step is to create the VACM Access table entry for the group name “mib2Read Write” with the access rights of read-write on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                   status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
    "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
```

```
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name. This would result in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The fourth step is to create the VACM Access table entry for the group name “mib2Read Only” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
    "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");
```

```
/* Clearing out write view. This will result in no Write Access.
 * Notify view is already being cleared out. So, there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name. This would result in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

This configuration can also be created by editing *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* as follows:

Add the following entry to *Snmpp\_Cfg\_Mib\_View* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_MIB\_VIEW\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```
{"mib2View", {1, 3, 6, 1, 2, 1}, 6, {0xFC}, 1, VACM_FAMILY_INCLUDED}
```

Add the following entry in *Snmpp\_Cfg\_Access* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_ACCESS\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```
{SNMP_CBSM_V2, "mib2ReadWriteNotify", "", VACM_CTXTMATCH_EXACT,
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", "mib2View"},

{SNMP_CBSM_V2, "mib2ReadWrite", "", VACM_CTXTMATCH_EXACT,
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", ""},

{SNMP_CBSM_V2, "mib2ReadOnly", "", VACM_CTXTMATCH_EXACT,
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "", ""}
```

Nucleus SNMP will enter into configuration mode when initialization from files failed. A real file system is required for successful initialization from file. During initialization from files Nucleus SNMP tried to restore its most recent state and when initialization from file failed Nucleus SNMP configured itself with the state defined in the configuration files.

## How to Use Access Entries - Security to Group Table

Now the next step is how to use these access entries. These access entries are only accessible by the group names assigned to them. For accessing group names, an entry is required to be added in Security to Group table.

An entry in the Security to Group table maps a combination of the security model and the security name to a group name. A group is a set of tuples of security model and security names. Different combination of security model and security names can be mapped to one or more groups. However, a particular combination of the security model and the security name can't be mapped to more than one group.

The Groups are designed to grant the security model independent access rights as the different combination of the security model and the security name can be mapped to single group.

For SNMPv2c, the security model would always be 2 (SNMP\_CBSM\_V2). You can only change the security name. So, the conclusion is that you can grant different access rights for different security names.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on VACM Security to Group table.

## Mapping Everything with Community Name in the Community Table

Nucleus SNMP derives the security name from the community name by accessing an entry from the Community table.

An entry in the Community table is uniquely identified by community index. There are two different fields for the community index and the community name. You will use the same values for both as it is a convention of Nucleus SNMP, advanced user may use different values.

This results in different access rights for the different communities. However, using the same security name for different communities would result in the same access rights for the different communities.

So, to complete the task of setting up the customized SNMPv2c securities, entries must be added to the Community table and the Security to Group table.

The following example illustrates a complete picture of designing access rights for SNMPv2c for the different communities.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on Community-Based Security Model.

### Example

This example is an extension of the example discussed in VACM Access table entries. In this example, you will create three communities named “mib2RdWtNotComm”, “mib2RdWtComm” and “mib2RdOnlyComm” with the access rights of read-write-notify, read-write and read-only on the MIB-2 Subtrees respectively.

The following are the steps involved in code example.

- Create a MIB View named “mib2View” that has access to the MIB-2 subtree.
- Create a VACM Access table entry for the group name “mib2ReadWrite Notify” that has read/write/notify access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadWrite” that has read/write access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadOnly” that has read-only access to the MIB View “mib2View” through SNMPv2c.
- Map the combination of the security model, CBSMv2, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.
- Map the combination of the security model, CBSMv2, and the security name “mib2RdWtSec” to the group named “mib2ReadWrite”.
- Map the combination of the security model, CBSMv2, and the security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.
- Create a Community table entry with the community name “mib2RdWtNot Comm” that is using the security name “mib2RdWtNotSec”.
- Create a Community table entry with the community name “mib2RdWtComm” that is using the security name “mib2RdWtSec”.
- Create a Community table entry with the community name “mib2RdOnlyComm” that is using the security name “mib2RdOnlySec”.

The first step is to create a MIB View named “mib2View” that have access to the MIB-2 subtree.

```
UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT vacm_mib_view;
STATUS                 status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
        sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);
```

The second step is to create the VACM Access table entry for the group name “mib2ReadWriteNotify” that has read/write/notify access to the MIB-2 Subtree using MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT          vacm_access_entry;
STATUS                      status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadWriteNotify");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. However, when you enable
 * prefix matching by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in access entry say
 * "ab" would also match context name like "abxxxx" and of
 * course with "ab". Please note that, an entry is required to
 * be there in Context table for the context name used in
 * Community table entry, otherwise access will not be granted.
 * But, context prefix in access entry is not require to
 * registered in Context table. The API routine for registering
 * a Context Name is VACM_InsertContext. An entry in context
 * table can also be added in configuration mode by modifying
 * Snmp_Cfg_Context_Names ("networking/snmp/snmp_cfg.c") and
 * VACM_CONTEXT_TBL_SIZE ("include/networking/snmp_cfg.h").
 */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif
```



```
/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The third step is to create the VACM Access table entry for the group name “mib2ReadWrite” with the access rights of read-write on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name. This would result in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The fourth step is to create the VACM Access table entry for the group name “mib2ReadOnly” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT          vacm_access_entry;
STATUS                       status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
        "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Clearing out write view. This will result in no Write Access.
 * Notify view is already being cleared out. So, there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name. This would result in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

You now have added the VACM Access table entries for the groups named “mib2ReadWriteNotify”, “mib2ReadWrite”, and “mib2ReadOnly” with the access rights according to their names. Now the next steps is to add relevant entries in the Security to Group table with the security model of SNMP\_CBSM\_V2 and the security names of “mib2RdWtNotSec”, “mib2RdWtSec” and “mib2RdOnlySec” respectively.

The fifth step is to map the combination of the security model, CBSMv2, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                      status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtNotSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWriteNotify");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
The sixth step is to map the combination of the security model, CBSMv2,
and the security name "mib2RdWtSec" to the group named "mib2ReadWrite".
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                      status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWrite");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

The seventh step is to map the combination of the security model, CBSMv2, and the security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                      status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdOnlySec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadOnly");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);

return (status);
```

You now have three security names “mib2RdWtNotSec”, “mib2RdWtSec”, and “mib2RdOnlySec” with access rights of read-write-notify, read-write and read-only on the MIB-2 subtree respectively.

In order to use these access rights, you are required to add entries in the Community table that uses these security names.

The eighth step is to create the Community table entry with the community name “mib2RdWtNotComm” that is using the security name “mib2RdWtNotSec”.

```
CBSM_COMMUNITY_STRUCT      community;
STATUS                      status;
UINT8                      engine_id[50];
UINT32                     engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *) community.cbsm_community_index,
       "mib2RdWtNotComm");
```

```
/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtNotComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtNotComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtNotSec");

/* Get the Engine Id.  SNMP determines Engine ID based upon
 * local IP Address.  A complex procedure is involved in
 * generation of this value.  So, it's better to retrieve the
 * value of Engine ID from Nucleus SNMP because this is
 * irrelevant here.  You do not want to lose the track.
 */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

The ninth step is to create the Community table entry with the community name “mib2RdWtComm” that is using the security name “mib2RdWtSec”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                    status;
UINT8                     engine_id[50];
UINT32                    engine_id_len;

/* Clear out the community structure. */
```

```
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdWtComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

The tenth step is to create the Community table entry with the community name “mib2RdOnlyComm” that is using the security name “mib2RdOnlySec”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
```

```
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdOnlyComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len = (UINT8)strlen("mib2RdOnlyComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdOnlyComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdOnlySec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return (status);

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "group1");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("group1");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
    /* Set the storage type. */
    community.cbsm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the status of community to active. */
    community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

This configuration can also be created by editing *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* as follows:

Add the following entry to *Snmpp\_Cfg\_Mib\_View* in *networking/snmp/snmp\_cfg.c* and update the value of *VACM\_MIB\_VIEW\_TBL\_SIZE* in *include/networking/snmp\_cfg.h* accordingly.

```
{"mib2View", {1, 3, 6, 1, 2, 1}, 6, {0xFC}, 1, VACM_FAMILY_INCLUDED}
```

Add the following entries in `Snm_Cfg_Access` in *networking/snmp/snmp\_cfg.c* and update the value of `VACM_ACCESS_TBL_SIZE` in *include/networking/snmp\_cfg.h* accordingly.

```
{SNMP_CBSM_V2, "mib2ReadWriteNotify", "", VACM_CTXTMATCH_EXACT,  
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", "mib2View"},  
{SNMP_CBSM_V2, "mib2ReadWrite", "", VACM_CTXTMATCH_EXACT,  
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "mib2View", ""},  
{SNMP_CBSM_V2, "mib2ReadOnly", "", VACM_CTXTMATCH_EXACT,  
SNMP_SECURITY_NOAUTHNOPRIV, "mib2View", "", ""}
```

Add the following entries to `Snm_Cfg_Sec2Groups` in *networking/snmp/snmp\_cfg.c* and update the value of `VACM_SEC2GRP_TBL_SIZE` in *include/networking/snmp\_cfg.h* accordingly.

```
{SNMP_CBSM_V2, "mib2RdWtNotSec", "mib2ReadWriteNotify"}  
{SNMP_CBSM_V2, "mib2RdWtSec", "mib2ReadWrite"}  
{SNMP_CBSM_V2, "mib2RdOnlySec", "mib2ReadOnly"}
```

And add the following entries to `CBSM_Community_Table` in *networking/snmp/snmp\_cfg.c* and update the value of `CBSM_MAX_COMMUNITIES` in *include/networking/snmp\_cfg.h* accordingly.

```
{"mib2RdWtNotComm", "mib2RdWtNotSec", "group1", ""}  
{"mib2RdWtComm", "mib2RdWtSec", "group1", ""}  
{"mib2RdOnlyComm", "mib2RdOnlySec", "group1", ""}
```

Nucleus SNMP will enter into configuration mode when initialization from files failed. A real file system is required for successful initialization from file. During initialization from files Nucleus SNMP tried to restore its most recent state and when initialization from file failed Nucleus SNMP configured itself with the state defined in the configuration files.

You now have three communities named “mib2RdWtNotComm”, “mib2RdWtComm”, “mib2RdOnlyComm” with access rights of read-write-notify, read-write and read-only on MIB-2 respectively.

## Additional Security by CBSM - Target Address Table

The discussion to this point illustrates that in SNMPv2c, you can set up different access rights for different communities. This may also lead to the false illusion that if an un-authorized person learns the name of a community, the person may have unauthorized access to the Nucleus SNMP Agent.

In addition to the previously discussed security checks SNMPv2c has one more check that is validating the IP Address from where SNMP request is received. The related table is called Target Address table.

Actually, the value of the transport tag in the Community table entry decides which entry is to use from the Target Address table. The entry in the Target Address is used to validate an IP



address. An entry in the Target Address table can have multiple authorized IP address or just a single IP address.

A request received from an un-authorized IP Address will not be processed by Nucleus SNMP Agent. You can configure SNMPv2c securities at the IP Address level as well.

## Example

Consider a case, that the security must be set up to allow the read-write and read-write-notify access is granted to the communities named “mib2RdWtNotComm” and “mib2RdWtComm”, and it is also required that these communities can only be accessible from a particular IP Address (say 192.168.0.14) and to grant read-only access to the community named “mib2RdOnlyComm” that is accessible by a pool of IP addresses of the from 192.168.0.0 to 192.168.0.255. This example is an extension of the previous example.

The following are the steps involved in the code example.

- Create a MIB View named “mib2View” that has access to the MIB-2 subtree.
- Create a VACM Access table entry for the group name “mib2ReadWriteNotify” that has read/write/notify access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadWrite” that has read/write access to the MIB View “mib2View” through SNMPv2c.
- Create a VACM Access table entry for the group name “mib2ReadOnly” that has read-only access to the MIB View “mib2View” through SNMPv2c.
- Map the combination of the security model, CBSMv2, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.
- Map the combination of the security model, CBSMv2, and the security name “mib2RdWtSec” to the group named “mib2ReadWrite”.
- Map the combination of the security model, CBSMv2, and security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.
- Create a Transport Address table entry named “HostRdWrNot” with transport tag “RdWrNotTag” and IP address 192.168.0.14.
- Create a Transport Address table entry named “HostRdWrNot” with the transport tag “HostRdOnlyTag” with the IP Address 192.168.0.14 and the IP Address mask 255.255.255.0, so that all the IP Addresses from 192.168.0.0 to 192.168.0.255 are included in it.
- Create a Community table entry with the community name “mib2RdWtNot Comm” that is using the security name “mib2RdWtNotSec” and the transport tag “RdWrNotTag”.
- Create a Community table entry with the community name “mib2RdWtComm” that is using the security name “mib2RdWtSec” and the transport tag “RdWrNotTag”.

- Create a Community table entry with the community name “mib2RdOnlyComm” that is using the security name “mib2RdOnlySec” and the transport tag “Host RdOnlyTag”.

The first step is to create a MIB view named “mib2View” that has access to the MIB-2 subtree.

```
UINT32                mib2_oid[] = {1, 3, 6, 1, 2, 1};
UINT8                 mib2_family_mask[] = {0xFC};
VACM_VIEWTREE_STRUCT  vacm_mib_view;
STATUS                 status;

memset(&vacm_mib_view, 0, sizeof(vacm_mib_view));
strcpy((CHAR *) (vacm_mib_view.vacm_view_name), "mib2View");
memcpy(vacm_mib_view.vacm_subtree, mib2_oid, sizeof(mib2_oid));
vacm_mib_view.vacm_subtree_len = 6;
memcpy(vacm_mib_view.vacm_family_mask, mib2_family_mask,
       sizeof(mib2_family_mask));
vacm_mib_view.vacm_mask_length = 1;
vacm_mib_view.vacm_family_type = VACM_FAMILY_INCLUDED;
#if (INCLUDE_MIB_VACM == NU_TRUE)
    vacm_mib_view.vacm_storage_type = SNMP_STORAGE_READONLY;
    vacm_mib_view.vacm_status = SNMP_ROW_ACTIVE;
#endif
status = VACM_InsertMibView(&vacm_mib_view);
```

The second step is to create the VACM Access table entry for the group name “mib2ReadWriteNotify” that has read/write/notify access to the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT    vacm_access_entry;
STATUS                 status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *) vacm_access_entry.vacm_group_name,
       "mib2ReadWriteNotify");

/* Set the value of context prefix. */
strcpy((CHAR *) vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. However, when you enable
 * prefix matching by setting the value of context_match to
 * VACM_CTXTMATCH_PREFIX, a context name in access entry say
 * "ab" would also match context name like "abxxxx" and of
 * course with "ab". Please note that, an entry is required to
 * be there in Context table for the context name used in
 * Community table entry, otherwise access will not be granted.
 * But, context prefix in access entry is not require to
 * registered in Context table. The API routine for registering
 * a Context Name is VACM_InsertContext. An entry in context
 * table can also be added in configuration mode by modifying
 * Snmp_Cfg_Context_Names ("networking/snmp/snmp_cfg.c") and
 * VACM_CONTEXT_TBL_SIZE ("include/networking/snmp_cfg.h").
 */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;
```

```
/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Make MIB2 Subtree notify-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "mib2View");

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set the Storage Type. */
    vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

    /* Set the entry status as 'active'. */
    vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The third step is to create the VACM Access table entry for the group name “mib2ReadWrite” with the access rights of read-write on the MIB-2 subtree using the MIB view “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT        vacm_access_entry;
STATUS                    status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
    "mib2ReadWrite");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");
```

```
/* Make MIB2 Subtree write-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "mib2View");

/* Clear the notify view name. This would result in no notify
 * access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the entry status as 'active'. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Add Access Entry to VACM Access table. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The fourth step is to create the VACM Access table entry for the group name “mib2ReadOnly” with the access rights of read-only on the MIB-2 subtree using the MIB View “mib2View” created in the first step.

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out access entry structure. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set the Group Name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "mib2ReadOnly");

/* Set the value of context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Exact context matching enabled. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set the value of security model. */
vacm_access_entry.vacm_security_model = SNMP_CBSM_V2;

/* Security Level in SNMPv2c is 'noAuthNoPriv'. */
vacm_access_entry.vacm_security_level =
    SNMP_SECURITY_NOAUTHNOPRIV;

/* Make MIB2 Subtree read-accessible. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "mib2View");

/* Clearing out write view. This will result in no Write Access.
 * Notify view is already being cleared out. So, there is
 * read-only access.
 */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

/* Clear the notify view name. This would result in no notify
```

```
* access.  
*/  
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");  
  
#if (INCLUDE_MIB_VACM == NU_TRUE)  
/* Set the Storage Type. */  
vacm_access_entry.vacm_storage_type = SNMP_STORAGE_READONLY;  
  
/* Set the entry status as 'active'. */  
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;  
#endif  
  
/* Add Access Entry to VACM Access table. */  
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

You now have added the VACM Access table entries for the groups named “mib2ReadWriteNotify”, “mib2ReadWrite”, and “mib2ReadOnly” with access rights according to their names. Now the next steps is to add relevant entries in the Security to Group table with the security model of SNMP\_CBSM\_V2 and the security name of “mib2RdWtNotSec”, “mib2RdWtSec”, and “mib2RdOnlySec” respectively.

The fifth step is to map the combination of the security model, CBSMv2, and the security name “mib2RdWtNotSec” to the group named “mib2ReadWriteNotify”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;  
STATUS                     status;  
  
/* Clear out the Security to Group table entry. */  
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));  
  
/* Set the Security Name. */  
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),  
       "mib2RdWtNotSec");  
  
/* Set the Security Model. */  
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;  
  
/* Set the Group Name. */  
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),  
       "mib2ReadWriteNotify");  
  
#if (INCLUDE_MIB_VACM == NU_TRUE)  
/* Set the Storage Type. */  
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;  
  
/* Set the Security to Group entry to 'active'. */  
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;  
#endif  
  
/* Adding entry to Security to Group table. */  
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

The sixth step is to map the combination of the security model, CBSMv2, and the security name “mib2RdWtSec” to the group named “mib2ReadWrite”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                      status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdWtSec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadWrite");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to Security to Group table. */
status = VACM_InsertGroup(&vacm_sec2group_entry);
```

The seventh step is to map the combination of the security model, CBSMv2, and the security name “mib2RdOnlySec” to the group named “mib2ReadOnly”.

```
VACM_SEC2GROUP_STRUCT      vacm_sec2group_entry;
STATUS                      status;

/* Clear out the Security to Group table entry. */
memset(&vacm_sec2group_entry, 0, sizeof(vacm_sec2group_entry));

/* Set the Security Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_security_name),
       "mib2RdOnlySec");

/* Set the Security Model. */
vacm_sec2group_entry.vacm_security_model = SNMP_CBSM_V2;

/* Set the Group Name. */
strcpy((CHAR *) (vacm_sec2group_entry.vacm_group_name),
       "mib2ReadOnly");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set the Storage Type. */
vacm_sec2group_entry.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set the Security to Group entry to 'active'. */
vacm_sec2group_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to Security to Group table. */
```

```
status = VACM_InsertGroup(&vacm_sec2group_entry);  
  
return (status);
```

You now have three security names “mib2RdWtNotSec”, “mib2RdWtSec”, and “mib2RdOnlySec” with the access rights of read-write-notify, read-write and read-only on the MIB-2 subtree respectively. To use these access rights, you are required to add entries in the Community table. You are able to have different access rights for different communities. Prior to creating the communities, you will create the Target Address table entries.

The eighth step is to create the Transport Address table entry named “HostRdWrNot” with the transport tag “RdWrNotTag” and the IP Address 192.168.0.14.

```
SNMP_TARGET_PARAMS_TABLE    target_param_entry;  
SNMP_TARGET_ADDRESS_TABLE   target_entry;  
STATUS                      status;  
  
/* Target Address on which you will grant read-write-notify  
 * access.  
 */  
UINT8      read_write_target_addr[IP_ADDR_LEN] = {192,168,0,14};  
  
/* Clear out the Target Address entry. */  
memset(&target_entry, 0, sizeof(target_entry));  
  
/* Set the name of Target table Entry. */  
strcpy(target_entry.snmp_target_addr_name, "HostRdWrNot");  
  
/* Set the Target Address. */  
memcpy(target_entry.snmp_target_addr_tAddress,  
        read_write_target_addr, IP_ADDR_LEN);  
  
/* Set the Target Address Family. */  
target_entry.snmp_target_addr_tfamily = NU_FAMILY_IP;  
  
/* Disable Target Address masking. So, that only a single  
 * address can be accessible by this Target Address entry.  
 */  
target_entry.snmp_ext_enabled = NU_FALSE;  
  
/* Set the transport tag it is used by Community table  
 * entry.  
 */  
strcpy((CHAR *)target_entry.snmp_target_addr_tag_list,  
        "RdWrNotTag");  
  
/* Set the length of transport tag. */  
target_entry.tag_list_len = (UINT32)strlen("RdWrNotTag");  
  
/* Set the name of Target Params table entry. You will create its  
 * entry in Target Params table later in this example.  
 */  
strcpy((CHAR *)target_entry.snmp_target_addr_params,  
        "RdWrNotTag-v2");
```

```
/* Set the length of Target Params. */
target_entry.params_len = (UINT32)strlen("RdWrNotTag-v2");

/* Set the transport domain. */
target_entry.snmp_target_addr_tDomain = SNMP_UDP;

/* Set the time out value. */
target_entry.snmp_target_addr_time_out = 1000;

/* Set the retry count. */
target_entry.snmp_target_addr_retry_count = 3;
#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Set the storage type. */
target_entry.snmp_target_addr_storage_type = SNMP_STORAGE_READONLY;

/* Set the Target Address entry as 'active'. */
target_entry.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Target Address entry. */
status = SNMP_Add_Target(&target_entry);

/* If you failed to add Target Address entry then return error
 * code.
 */
if (status != NU_SUCCESS)
    return status;

/* Clear out Target Params table entry. */
memset(&target_param_entry, 0, sizeof(target_param_entry));

/* Set the Target Params table entry name. */
strcpy(target_param_entry.snmp_target_params_name,
        "RdWrNotTag-v2");

/* Set the Message Processing Model. */
target_param_entry.snmp_target_params_mp_model = SNMP_VERSION_V2;

/* Set the Security Name. */
strcpy(target_param_entry.snmp_target_params_security_name,
        "mib2RdWtNotSec");

/* Set the security model. */
target_param_entry.snmp_target_params_security_model =
        SNMP_CBSM_V2;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Set the Target Params entry's storage type. */
target_param_entry.snmp_target_params_storage_type =
        SNMP_STORAGE_READONLY;

/* Set the target para table entry as 'active'. */
target_param_entry.snmp_target_params_row_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry in Target Params table. */
status = SNMP_Add_Params(&target_param_entry);
```



The ninth step is to create the Transport Address table entry named “HostRdWrNot” with the transport tag “HostRdOnlyTag”, the IP Address of 192.168.0.14, and the IP Address Mask 255.255.255.0, so that all the IP Addresses from 192.168.0.0 to 192.168.0.255 are included.

```
SNMP_TARGET_PARAMS_TABLE    target_param_entry;
SNMP_TARGET_ADDRESS_TABLE    target_entry;
STATUS                       status;

/* Target Address on which you will grant access read-only access.
 */
UINT8      read_only_target_addr[IP_ADDR_LEN] = {192,168,0,0};

/* You want to grant read-only access to pool of IP addresses
 * from 192.168.0.0 to 192.168.0.255. Zero in address mask
 * represents a wild card. Now, 192.168.0.14 is included in
 * both. Now, access rights for Target Address 192.168.0.14
 * would be determined by the community name it uses in its
 * requests.
 */
UINT8      read_only_target_mask[IP_ADDR_LEN] = {255, 255, 255, 0};

/* Clear out the Target Address entry. */
memset(&target_entry, 0, sizeof(target_entry));

/* Set the name of Target table entry. */
strcpy(target_entry.snmp_target_addr_name, "HostRdWrNot");

/* Set the Target Address. */
memcpy(target_entry.snmp_target_addr_tAddress,
        read_only_target_addr, IP_ADDR_LEN);

/* Set the Target Address Family. */
target_entry.snmp_target_addr_tfamily = NU_FAMILY_IP;

/* Enable Target Address masking. So, that you can use a pool
 * of IP addresses with this Target Address entries.
 */
target_entry.snmp_ext_enabled = NU_TRUE;

/* Set the Target Address mask. */
memcpy(target_entry.snmp_target_addr_tmask,
        read_only_target_mask, IP_ADDR_LEN);

/* Set the transport tag it is used by Community table
 * entry.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_tag_list,
        "HostRdOnlyTag");

/* Set the length of transport tag. */
target_entry.tag_list_len = (UINT32)strlen("RdWrNotTag");

/* Set the name of Target Params table entry. You will create its
 * entry in Target Params table later in this example.
 */
strcpy((CHAR *)target_entry.snmp_target_addr_params,
        "ParamRdOnly-v2");
```

```
/* Set the length of Target Params. */
target_entry.params_len = (UINT32)strlen("RdWrNotTag-v2");

/* Set the transport domain. */
target_entry.snmp_target_addr_tDomain = SNMP_UDP;

/* Set the time out value. */
target_entry.snmp_target_addr_time_out = 1000;

/* Set the retry count. */
target_entry.snmp_target_addr_retry_count = 3;

/* Set the maximum message size. */
target_entry.snmp_target_addr_mms = 500;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Set the storage type. */
target_entry.snmp_target_addr_storage_type = SNMP_STORAGE_READONLY;

/* Set the Target Address entry as 'active'. */
target_entry.snmp_target_addr_row_status = SNMP_ROW_ACTIVE;
#endif

/* Add the Target Address entry. */
status = SNMP_Add_Target(&target_entry);

/* If you failed to add Target Address entry then return error
 * code.
 */
if (status != NU_SUCCESS)
    return status;

/* Clear out Target Params table entry. */
memset(&target_param_entry, 0, sizeof(target_param_entry));

/* Set the Target Params table entry name. */
strcpy(target_param_entry.snmp_target_params_name,
        "ParamRdOnly-v2");

/* Set the Message Processing Model. */
target_param_entry.snmp_target_params_mp_model = SNMP_VERSION_V2;

/* Set the Security Name. */
strcpy(target_param_entry.snmp_target_params_security_name,
        "mib2RdOnlySec");

/* Set the security model. */
target_param_entry.snmp_target_params_security_model = SNMP_CBSM_V2;

#if (INCLUDE_MIB_TARGET == NU_TRUE)
/* Set the Target Params entry's storage type. */
target_param_entry.snmp_target_params_storage_type =
    SNMP_STORAGE_READONLY;

/* Set the target para table entry as 'active'. */
target_param_entry.snmp_target_params_row_status = SNMP_ROW_ACTIVE;
#endif
```

```
/* Adding entry in Target Params table. */
status = SNMP_Add_Params(&target_param_entry);
```

The tenth step is to create the Community table entry with the community name “mib2RdWtNotComm” that is using the security name “mib2RdWtNotSec” and the transport tag “RdWrNotTag”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                   engine_id[50];
UINT32                  engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index,
       "mib2RdWtNotComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtNotComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtNotComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtNotSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. Nucleus SNMP would search
 * all the entries with this transport tag. This really
 * helps in mapping multiple IP address pools to a
 * single community.
 */
strcpy((CHAR *)community.cbsm_transport_tag, "RdWrNotTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("RdWrNotTag");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
```

```
/* Set the storage type. */
community.cbsm_storage_type = SNMP_STORAGE_READONLY;

/* Set the status of community to active. */
community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

The eleventh step is to create the Community table entry with the community name “mib2RdWtComm” that is using the security name “mib2RdWtSec” and the transport tag “RdWrNotTag”.

```
CBSM_COMMUNITY_STRUCT    community;
STATUS                   status;
UINT8                    engine_id[50];
UINT32                   engine_id_len;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdWtComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdWtComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdWtComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdWtSec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "RdWrNotTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("RdWrNotTag");

#if (INCLUDE_MIB_CBSM == NU_TRUE)
```

```
/* Set the storage type. */
community.cbsm_storage_type = SNMP_STORAGE_READONLY;

/* Set the status of community to active. */
community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

The eleventh and the final step is to create the Community table entry with the community name “mib2RdOnlyComm” that is using the security name “mib2RdOnlySec” and the transport tag “HostRdOnlyTag”.

```
CBSM_COMMUNITY_STRUCT    community;
UINT8                    engine_id[50];
UINT32                   engine_id_len;
STATUS                   status;

/* Clear out the community structure. */
memset(&community, 0, sizeof(community));

/* Set the Community Index. */
strcpy((CHAR *)community.cbsm_community_index, "mib2RdOnlyComm");

/* Set the Community Index Length. */
community.cbsm_community_index_len =
    (UINT8)strlen("mib2RdOnlyComm");

/* Set the Community Name. */
strcpy((CHAR *)community.cbsm_community_name, "mib2RdOnlyComm");

/* Set the Security Name. */
strcpy((CHAR *)community.cbsm_security_name, "mib2RdOnlySec");

/* Get the Engine Id. */
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);

/* If you failed to get Engine ID return error code. */
if (status != NU_SUCCESS)
    return status;

/* Set the Community Engine ID. */
memcpy(community.cbsm_engine_id, engine_id, engine_id_len);

/* Set the Community Engine ID length. */
community.cbsm_engine_id_len = (UINT8)engine_id_len;

/* Set the context name. */
strcpy((CHAR *)community.cbsm_context_name, "");

/* Set the transport tag. */
strcpy((CHAR *)community.cbsm_transport_tag, "HostRdOnlyTag");

/* Set the transport tag length. */
community.cbsm_transport_tag_len = strlen("HostRdOnlyTag");
```

```
#if (INCLUDE_MIB_CBSM == NU_TRUE)
/* Set the storage type. */
community.cbsm_storage_type = SNMP_STORAGE_READONLY;

/* Set the status of community to active. */
community.cbsm_status = SNMP_ROW_ACTIVE;
#endif

/* Adding entry to the Community table. */
status = CBSM_Add_Community(&community);
```

This configuration cannot be created by setting the values in *networking/snmp/snmp\_cfg.c* or *include/networking/snmp\_cfg.h* because Nucleus SNMP does not provide an interface to create Transport IP Address pools in the *Snmp\_Cfg\_Tgr\_Addr\_Tbl*.

## SNMPv3 Security Configurations

After reading the Security Configurations of SNMPv1, SNMPv2c, you understand that the Security Model after security validation outputs the following parameters that is used as input to VACM.

- security name
- context name
- security model
- security level

The message processing model has to provide the following parameters as input to VACM.

- OID of the object
- access type (read-write-notify)

The VACM is also required to be configured for SNMPv3.

The VACM Module's security configuration has been discussed under the Security Configurations of SNMPv1 and SNMPv2c in this chapter.

SNMPv3 uses the security model of USM (User-based Security Model). There are two Authentication Protocols supported within the USM module that are HMAC-MD5 and HMAC-SHA-1.

Refer to [“Advanced SNMP Topics”](#) on page 1693 for more information on User-based Security Model.

To configure SNMPv3 security, you need to add a user in the targeted User. Then you are required to configure VACM module as required.

## Example

In the following example, a user was created with user name and security name “snmpv3authSec”, authentication password “anythingforsec”, privacy password “privacypass” that has authPriv access of read-write-notify over MIB View “internet” and AuthNoPriv access of read-only over MIB View “internet”.

Perform the following steps in the code example:

- Create a USM User with user and security name “snmpv3authSec”. The Authentication Protocol used by this user would be HMAC-MD5 with authentication password “anythingforsec”. The privacy password used is CBC-DES with the privacy password of “privacypass”.
- Map the combination of the security name “snmpv3authSec” and the security model, USM, to group name “snmpv3authGrp”.
- Create a VACM Access Table entry for the group name “snmpv3authGrp” with the security level of AuthPriv that has access rights of read/right/notify over the MIB View “internet” (A View that is defined in default SNMP configuration).
- Create a VACM Access table entry for the group name “snmpv3authGrp” with the security level of AuthNoPriv that has access rights of read-only over the MIB View “internet” (A View that is defined in default SNMP configuration).

The first step is to create a USM User with user and security name “snmpv3authSec”. The Authentication Protocol used by this user would be HMAC-MD5 with authentication password “anythingforsec”. The privacy password used is CBC-DES with the privacy password of “privacypass”.

```
STATUS                status;
UINT8                 engine_id[50];
UINT32                engine_id_len;
USM_USERS_STRUCT      user;
USM_AUTH_PROT_STRUCT  *auth_prot;

#if (USM_PASSWORD_2_KEY == NU_TRUE)
    /* Authentication password. */
    UINT8 auth_password[] = "anythingforsec";

    /* Privacy password. */
    UINT8 privacy_passowrd[] = "privacypass";

#else

    /* Key Generated by pass2key utility for auth_password.
     * Command dos prompt is as follows:
     *     cmd> pass2key M anythingforsec
     * pass2key.exe is shipped with Nucleus SNMP.
     */
    UINT8 auth_password[] =
        {0x79, 0x29, 0xFF, 0xF5,
         0x58, 0x61, 0x35, 0x0D,
```

```
                                0x66, 0x96, 0x6D, 0x2C,  
                                0x31, 0x95, 0x8E, 0x5B};  
  
/* Key Generated by pass2key utility for privacy_passowrd.  
 * Command dos prompt is as follows:  
 *      cmd> pass2key M privacypass  
 */  
UINT8 privacy_passowrd[] =  
    {0xB9, 0x33, 0xAA, 0x10,  
     0xF5, 0x8A, 0x10, 0x1A,  
     0xF4, 0xA8, 0x79, 0x4E,  
     0x32, 0xE0, 0x4B, 0xE7};  
  
#endif  
  
/* Get SNMP Engine ID. */  
status = SNMP_Get_Engine_ID(engine_id, &engine_id_len);  
  
/* If you failed to get SNMP Engine ID then return error code. */  
if (status != NU_SUCCESS)  
    return status;  
  
/* Clear out the user entry structure. */  
memset(&user, 0, sizeof(user));  
  
/* Set SNMP Engine ID in USM user entry structure. */  
memcpy(user.usm_user_engine_id, engine_id, engine_id_len);  
  
/* Set SNMP Engine ID length in USM user entry structure. */  
user.usm_user_engine_id_len = (UINT8)engine_id_len;  
  
/* Set user name. */  
strcpy((CHAR *) (user.usm_user_name), "snmpv3authSec");  
  
/* Set security name. */  
strcpy((CHAR *) user.usm_security_name, "snmpv3authSec");  
  
/* Set Authentication Protocol to MD5. */  
user.usm_auth_index = USM_MD5;  
  
/* Get handle to Authentication Protocol MD5. */  
auth_prot = USM_Lookup_Auth_Prot(user.usm_auth_index);  
  
/* If you failed to get handle to Authentication  
 * Protocol return error code.  
 */  
if ((auth_prot == NU_NULL) ||  
    (auth_prot->usm_password_cb == NU_NULL))  
    return -1;  
  
/* Get the Authentication Key. */  
#if (USM_PASSWORD_2_KEY == NU_TRUE)  
    auth_prot->usm_password_cb(auth_password,  
                               (UINT8)(strlen((CHAR *) auth_password)),  
                               user.usm_auth_key);  
#else  
    auth_prot->usm_password_cb(auth_password, 16,  
                               user.usm_auth_key);  
#endif
```



```
#endif

/* Update the user own authentication key with
 * authentication key. This actual key used
 * when authentication.
 */
memcpy(user.usm_own_auth_key, user.usm_auth_key, 16);

/* Set Privacy Protocol to DES. */
user.usm_priv_index = USM_DES;

/* Get the Privacy Key. */
#if (USM_PASSWORD_2_KEY == NU_TRUE)
    auth_prot->usm_password_cb(privacy_passowrd,
                                (UINT8)(strlen((CHAR *)privacy_passowrd)),
                                user.usm_priv_key);
#else
    auth_prot->usm_password_cb(privacy_passowrd, 16,
                                user.usm_priv_key);
#endif

/* Update the user owned privacy key.
 * This actual key that is used.
 */
memcpy(user.usm_own_priv_key, user.usm_priv_key, 16);

#if (INCLUDE_MIB_USM == NU_TRUE)
    /* Set storage type to permanent. */
    user.usm_storage_type = SNMP_STORAGE_READONLY;

    /* Set row status as active. */
    user.usm_status = SNMP_ROW_ACTIVE;
#endif

/* Add this entry to the USM User table. */
status = USM_Add_User(&user);
```

The second step is to map the combination of the security name “snmpv3authSec” and the security model, USM, to the group name “snmpv3authGrp”.

```
VACM_SEC2GROUP_STRUCT    vacm_sec2grp;
STATUS                   status;

/* Clearout security to group table entry. */
memset(&vacm_sec2grp, 0, sizeof(vacm_sec2grp));

/* Set value to group name. */
strcpy((CHAR *)vacm_sec2grp.vacm_group_name, "snmpv3authGrp");

/* Set value to security name. */
strcpy((CHAR *)vacm_sec2grp.vacm_security_name, "snmpv3authSec");

/* Set security model to USM. */
vacm_sec2grp.vacm_security_model = SNMP_USM;

#if (INCLUDE_MIB_VACM == NU_TRUE)
    /* Set storage type. */
```

```
vacm_sec2grp.vacm_storage_type = SNMP_STORAGE_READONLY;

/* Set row status as 'active'. */
vacm_sec2grp.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Insert Security to Group entry. */
status = VACM_InsertGroup(&vacm_sec2grp);
```

The third step is to create a VACM Access table entry for the group name “snmpv3auth Grp” with the security level of AuthPriv that has access rights of read/right/notify over the MIB View “internet” (A View that is defined in default SNMP configuration).

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out the access entry. */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set context match 'exact'. */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set context prefix. */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Set group name. */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "snmpv3authGrp");

/* Set security level. */
vacm_access_entry.vacm_security_level = SNMP_SECURITY_AUTHPRIV;

/* Set security model. */
vacm_access_entry.vacm_security_model = SNMP_USM;

/* Set notify view. */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "internet");

/* Set read view. */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "internet");

/* Set write view. */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "internet");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set storage type. */
vacm_access_entry.vacm_storage_type = SNMP_ROW_ACTIVE;

/* Set row status as active. */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Insert access entry. */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

The fourth step is to create a Access Entry for the group name “snmpv3authGrp” with the security level of AuthNoPriv that has access rights of read-only over the MIB View “internet” (A View that is defined in default SNMP configuration)

```
VACM_ACCESS_STRUCT      vacm_access_entry;
STATUS                  status;

/* Clear out the access entry.  */
memset(&vacm_access_entry, 0, sizeof(vacm_access_entry));

/* Set context match 'exact'.  */
vacm_access_entry.vacm_context_match = VACM_CTXTMATCH_EXACT;

/* Set context prefix.  */
strcpy((CHAR *)vacm_access_entry.vacm_context_prefix, "");

/* Set group name.  */
strcpy((CHAR *)vacm_access_entry.vacm_group_name,
       "snmpv3authGrp");

/* Set security level.  */
vacm_access_entry.vacm_security_level = SNMP_SECURITY_AUTHNOPRIV;

/* Set security model.  */
vacm_access_entry.vacm_security_model = SNMP_USM;

/* Set notify view.  */
strcpy((CHAR *)vacm_access_entry.vacm_notify_view, "");

/* Set read view.  */
strcpy((CHAR *)vacm_access_entry.vacm_read_view, "internet");

/* Set write view.  */
strcpy((CHAR *)vacm_access_entry.vacm_write_view, "");

#if (INCLUDE_MIB_VACM == NU_TRUE)
/* Set storage type.  */
vacm_access_entry.vacm_storage_type = SNMP_ROW_ACTIVE;

/* Set row status as active.  */
vacm_access_entry.vacm_status = SNMP_ROW_ACTIVE;
#endif

/* Insert access entry.  */
status = VACM_InsertAccessEntry(&vacm_access_entry);
```

This configuration can also be created by editing *networking/snmp/snmp\_cfg.c* and *include/networking/snmp\_cfg.h* as follows:

Add the following entry in [Usm\\_User\\_Table](#) in *snmp\_cfg.c* and update the value of `USM_MAX_USER_USERS` in *snmp\_cfg.h*.

```
/* User Name, Auth Algo, Auth Password, Priv Algo,
 * Priv Password */
{"snmpv3authSec", USM_MD5, "anythingforsec", USM_DES, "privacypass"}
```

When the value of `USM_PASSWORD_2_KEY` is `NU_FALSE` then Authentication and privacy keys are used instead of passwords in the entry of [Usm\\_User\\_Table](#).

Add the following entry in `Snm_Cfg_Sec2Groups` in *snmp\_cfg.c* and update the value of `VACM_SEC2GRP_TBL_SIZE` in *snmp\_cfg.h*.

```
/* Security model, Security name, Group name. */
{SNMP_USM, "snmpv3authSec", "snmpv3authGrp"}
```

Add the following entries in `Snm_Cfg_Access` in *snmp\_cfg.c* and update the value of `VACM_ACCESS_TBL_SIZE` in *snmp\_cfg.h*.

```
/* Security model, Group name, Context prefix, Security level,
   Context match, Read view, Write view, Notify view. */
{SNMP_USM, "snmpv3authGrp", "", SNMP_SECURITY_AUTHPRIV,
 VACM_CTXTMATCH_EXACT, "internet", "internet", "internet"}

{SNMP_USM, "snmpv3authGrp", "", SNMP_SECURITY_AUTHNOPRIV,
 VACM_CTXTMATCH_EXACT, "internet", "", ""}
```

Nucleus SNMP will enter into configuration mode when initialization from files failed. A real file system is required for successful initialization from file. During initialization from files Nucleus SNMP tried to restore its most recent state and when initialization from file failed Nucleus SNMP configured itself with the state defined in the configuration files.

## Enhancements and API Changes

### Nucleus SNMP 2.1

While Nucleus SNMP 2.1 contains many improvements over the prior versions of Nucleus SNMP, an attempt was made to modify the existing API as little as possible. This section describes the changes that were necessary.

#### New SNMP 2.1 Initialization Sequence

The initialization of Nucleus SNMP was changed. Prior to Nucleus SNMP 2.1, the application made calls to `SNMP_Initialize` and `SNMP_System_Group_Initialize` to perform initialization. Now, the application must not call these routines. Next, the service call to `SNMP_Configuration` is required. If `cfig_hostid` in *networking/snmp/snmp\_cfg.c* is not configured or the application makes use of DHCP to determine the IP Address at run-time. A service call to `NU_SNMP_Set_Host_Id` is also required prior to `SNMP_Configuration`. Prior to the calls to `SNMP_Configuration` and `NU_SNMP_Set_Host_Id` Nucleus FAL, Nucleus NET, and the interface devices must have been initialized.

```
NU_IOCTL_OPTION device_ip;

/* Initialize FAL, NET and system interfaces. */
```

```
/* Point to the name of the device that you wish to know the IP
 * address of. */
device_ip.s_optval = (UINT8*) DEMOI_Device_Name;

/* Call NU_Ioctl to get the IP address. */
if (NU_Ioctl (IOCTL_GETIFADDR, &device_ip,
             sizeof(NU_IOCTL_OPTION)) != NU_SUCCESS)
{
    printf ("DEMOI_Obtain_Ip: Failed to obtain local IP Address.\n");
    DEMOI_Exit (__LINE__);
}

/* Copy the retrieved IP address. */
memcpy (DEMOI_Local_Ip, device_ip.s_ret.s_ipaddr, 4);

/* Set the local IP for the SNMP Engine ID. */
NU_SNMP_Set_Host_Id (DEMOI_Local_Ip);

/* SNMP modules may need to be configured, if initialization was
 * not successful (possible because SNMP is being run for the
 * first time).
 */
SNMP_Configuration ();
```

## New SNMP 2.1 API Service Calls

- NU\_SNMP\_Initialize
- SNMP\_Configuration
- NU\_SNMP\_Set\_Host\_Id
- SNMP\_Add\_In\_Notifications
- AgentGetAuthenTraps
- AgentSetAuthenTraps
- AgentSetColdTraps

## Removed SNMP 2.1 API Service Calls

- AddComm
- AddHost
- AgentSendTrap

Nucleus SNMP v 2.1 does not provide work-around for AddComm and AddHost API Service Calls.

## Sending Notifications - AgentSendTrap work-around

The example for sending out notifications is as follows:

```
/* Allocating the memory to notification structure to send
starting trap. */
if (NU_Allocate_Memory(&System_Memory, &snmp_notification,
                      sizeof(SNMP_NOTIFY_STRUCT),
                      NU_NO_SUSPEND) == NU_SUCCESS)
{
    /* Clear the structure. */
    UTL_Zero(snm_notification, sizeof(SNMP_NOTIFY_STRUCT));

    /* Copy the cold start OID. */
    memcpy(snm_notification->OID.notification_oid,
           cold_start_oid,
           sizeof(UINT32) * SNMP_TRAP_OID_LEN);

    snm_notification->OID.oid_len = SNMP_TRAP_OID_LEN;

    /* Send the notification. */
    SNMP_Add_In_Notifications(snm_notification);

    /* de-allocating memory */
    NU_Deallocate_Memory((VOID*) snm_notification);
}
```

## Nucleus SNMP 2.2

While Nucleus SNMP 2.2 contains many improvements over the prior versions of Nucleus SNMP, an attempt was made to modify the existing API as little as possible. This section describes the changes that were necessary.

### New SNMP 2.2 API Service Calls

- [SNMP\\_Get\\_Notification\\_Ptr](#)
- [SNMP\\_Notification\\_Ready](#)
- [SNMP\\_Get\\_Engine\\_ID](#)
- [CBSM\\_Add\\_Community](#)
- [CBSM\\_Find\\_Community\\_Index](#)
- [CBSM\\_Remove\\_Community](#)
- [CBSM\\_Save\\_Community](#)
- [SNMP\\_Add\\_To\\_Notify\\_Tbl](#)
- [SNMP\\_Find\\_Notify\\_Entry](#)
- [SNMP\\_Remove\\_From\\_Notify\\_Table](#)

- [SNMP\\_Add\\_To\\_Profile\\_Tbl](#)
- [SNMP\\_Find\\_Profile\\_Entry](#)
- [SNMP\\_Remove\\_From\\_Profile\\_Table](#)
- [SNMP\\_Add\\_To\\_Filter\\_Tbl](#)
- [SNMP\\_Find\\_Filter\\_Entry](#)
- [SNMP\\_Remove\\_From\\_Filter\\_Table](#)
- [SNMP\\_Add\\_Target](#)
- [SNMP\\_Find\\_Target](#)
- [SNMP\\_Remove\\_From\\_Tgr\\_Table](#)
- [SNMP\\_Add\\_Params](#)
- [SNMP\\_Find\\_Params](#)
- [SNMP\\_Remove\\_From\\_Params\\_Table](#)
- [USM\\_Add\\_User](#)
- [USM\\_Lookup\\_Users](#)
- [USM\\_Remove\\_User](#)
- [USM\\_Save\\_User](#)
- [USM\\_Lookup\\_Priv\\_Prot](#)
- [USM\\_Lookup\\_Auth\\_Prot](#)
- [VACM\\_Add\\_Context](#)
- [VACM\\_Search\\_Context](#)
- [VACM\\_Remove\\_Context](#)
- [VACM\\_InsertGroup](#)
- [VACM\\_Search\\_Group](#)
- [VACM\\_Remove\\_Group](#)
- [VACM\\_Save\\_Group](#)
- [VACM\\_InsertAccessEntry](#)
- [VACM\\_Search\\_AccessEntry](#)
- [VACM\\_Remove\\_AccessEntry](#)
- [VACM\\_Save\\_Access](#)

- [VACM\\_InsertMibView](#)
- [VACM\\_Search\\_MibView](#)
- [VACM\\_Remove\\_MibView](#)
- [VACM\\_Save\\_View](#)

## Removed SNMP 2.2 API Service Calls

- [SNMP\\_Add\\_In\\_Notifications](#)

## Sending Notifications - [SNMP\\_Add\\_In\\_Notifications](#) work-around

The following is example for sending out notifications or traps.

```
/* Handle to the notification request structure. */
SNMP_NOTIFY_REQ_STRUCT *snmp_notification;

/* OID of notification. */
UINT32 notification_oid[] =
    {1, 3, 6, 1, 2, 1, 55, 2, 0, 1};

/* OID of first binded object. */
UINT32 binded_object1_oid[] =
    {1, 3, 6, 1, 2, 1, 55, 1, 5, 1, 2, 1, 1};

/* OID of second binded object. */
UINT32 binded_object2_oid [] =
    {1, 3, 6, 1, 2, 1, 55, 1, 5, 1, 10};

/* If you successfully got the handle to the notification. */
if (SNMP_Get_Notification_Ptr (&snmp_notification) ==
    NU_SUCCESS)
{
    /* Clear out the notification structure. */
    memset (snmp_notification, 0,
            sizeof(SNMP_NOTIFY_REQ_STRUCT));

    /* Update the notification OID. */
    memcpy (snmp_notification->OID.notification_oid,
            notification_oid,
            sizeof(notification_oid));

    /* Update the notification OID length. */
    snmp_notification->OID.oid_len =
        (sizeof(notification_oid) / sizeof(UINT32));

    /* Set first binded object. */
    memcpy (snmp_notification->snmp_object_list[0].Id,
            binded_object1_oid,
            sizeof(binded_object1_oid));

    /* Set OID length. */
    snmp_notification->snmp_object_list[0].IdLen =
```



```
        (sizeof(binded_object1_oid) / sizeof(UINT32));

/* Set second binded object. */
memcpy (snmp_notification->snmp_object_list[1].Id,
        binded_object2_oid,
        sizeof(binded_object2_oid));

/* Set OID length. */
snmp_notification->snmp_object_list[1].IdLen =
    (sizeof(binded_object2_oid) / sizeof(UINT32));

/* Update the number of bound objects. */
snmp_notification->snmp_object_list_len = 2;

/* Send the notification. */
SNMP_Notification_Ready (snmp_notification);
}
```

## Adding Communities and Target Addresses - AddComm and AddHost work-around

Nucleus SNMP 1.x uses AddComm and AddHost API Service Calls for security configuration. Nucleus SNMP 2.1 does not provide any work-around for these APIs.

In prior versions, the community is referred to by a community name and the access allowed to the community is constant. That is, if a community is write-accessible then it has write access over all MIB objects. This is similar to read and notify access. Now, you can better configure the access rights. The access rights can be configured in way that a community can have write-access over a group of MIB objects and do not have write-access over other MIB objects. Similarly, read-access and notify-access can also be managed.

In prior versions, the community table entry does not know the SNMP Manager IP Addresses from which it is accessible. The Host table entry defines the IP Address of SNMP Manager and a community that is accessible to that SNMP Manager. Now, the Community table entry has a field called transport tag. The transport tag contains a string. On the other end, a Target Address (mapping of Host Table) has a field for the transport tags list separated by the space and tab characters. When the transport tag value is an empty string in the Community table entry, it is accessible from any SNMP Manager having any IP Address. When it contains a non-empty string value, it is accessible only through those IP Addresses defined by the Target Address table entries with this transport tag in their lists.

Refer to the [Advanced SNMP Topics](#) for more information on Community-Base Security Model (CBSM). Refer to the [SNMPv1 Security Configurations](#) for source examples on how to configure Nucleus SNMP.

## Nucleus SNMP 2.3

Nucleus SNMP 2.3 contains many improvements over the prior versions of Nucleus SNMP. This section describes the new API services in Nucleus SNMP 2.3.

## New SNMP 2.3 Initialization Sequence

The initialization sequence that is used in prior version of Nucleus SNMP is also valid for Nucleus SNMP 2.3. However, Nucleus SNMP 2.3 also provides an API Service Call `NU_SNMP_Set_Host_Id6` that may be used instead of `NU_SNMP_Set_Host_Id` in order to initialize Engine ID with IPv6 address.

```
NU_IOCTL_OPTION    device_ip;

/* Initialize FAL, NET and system interfaces. */

/* Point to the name of the device that you wish to know the IP
   address of. */
device_ip.s_optval = (UINT8*) DEMOI_Device_Name;

/* Call NU_Ioctl to get the IP address. */
NU_Ioctl (SIOCGIFADDR_IN6, &device_ip, sizeof(NU_IOCTL_OPTION));

/* Set the local IP for the SNMP Engine ID. */
NU_SNMP_Set_Host_Id6 (device_ip.s_ret.s_ipaddr);

/* SNMP modules may need to be configured, if initialization was
 * not successful (possible because SNMP is being run for the
 * first time).
 */
SNMP_Configuration ();
```

## New SNMP 2.3 Service Calls

- `NU_SNMP_Set_Host_Id6`
- `SNMP_Mib_Register`

## RMON Release Dependency Removed

Prior to the Nucleus SNMP 2.3 release, Nucleus RMON was always released simultaneously with Nucleus SNMP and shared the same reference manual. As of the Nucleus SNMP 2.3 release, Nucleus RMON is released separately from Nucleus SNMP and will have its own reference manual.

## Nucleus SNMP 2.4

Nucleus SNMP 2.4 contains many bug fixes. The bugs have been identified by the SNMP SilverCreek Test Suite and have been done in MIB-II and various other components of Nucleus SNMP. No new features have been added in Nucleus SNMP 2.4 apart from a single API to unregister MIBs at run-time.

This section describes the new API services in Nucleus SNMP 2.4.

## New SNMP 2.4 Service Calls

- `SNMP_Mib_Unregister`











































































































































































































































































































































































































































# Chapter 10

## Extended Protocol (XPROT)

---

### XPROT Overview

The Nucleus Extended Protocol Package (XPROT) contains an RFC-compliant FTP Server, FTP Client, Telnet Server, Telnet Client and TFTP Server. All of the above protocols can execute in IPv4-only, IPv6-only or in dual-stack IPv4/IPv6.

To enable this, include *os/include/networking/nu\_networking.h* in your application.

---

#### Warning



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

### FTP Server

The Nucleus FTP Server API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC-compliant FTP Server implementation over IPv4, IPv6 or dual-stack IPv4/IPv6 and serve as an example for building custom servers.

### FTP Client

The Nucleus FTP Client API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC-compliant framework upon which an FTP Client application can be built over IPv4, IPv6 or dual-stack IPv4/IPv6. They are designed to allow for a variety of different client implementations, ranging from an interactive user-oriented client to an embedded special-use client.

### Telnet Server

The Nucleus Telnet Server API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC-compliant framework upon which a simple Telnet Server application can be built for an embedded system over IPv4, IPv6 or dual-stack IPv4/IPv6. For example, a Telnet debugger server can be built as a task in an application. The Telnet facility could then be used on another computer to connect to this debugger server and retrieve Nucleus debugger information for the application.

## Telnet Client

The Nucleus Telnet Client API is a set of functions for use with Nucleus PLUS in conjunction with Nucleus NET. These functions provide an RFC-compliant framework upon which a simple Telnet client application can be built for an embedded system over IPv4, IPv6 or dual-stack IPv4/IPv6. For example, a Telnet client can be built as a task in an application. The Telnet client could be used to connect to a remote server and run the necessary commands on the server.

## Trivial File Transfer Protocol Server

The Nucleus Trivial File Transfer Protocol (TFTP) Server is an implementation of an RFC-compliant TFTP server designed to run on top of the Nucleus NET protocol stack over IPv4, IPv6 or dual-stack IPv4/IPv6 and the Nucleus PLUS RTOS. TFTP is, as its name implies, a very simple file transfer protocol. Using the Nucleus TFTP Server, you can implement the TFTP Server and have it executing as a separate task than the user's application. The Nucleus TFTP Server is compatible with RFC 2347 compliant and non-compliant TFTP Client applications.

## FTP Package

Nucleus FTP Server and Client are a part of the Nucleus XPROT. The Nucleus FTP Server and Client can operate over IPv4, IPv6 or dual-stack IPv4/IPv6 protocols. The Nucleus FTP Server provides file transfer services.

The Nucleus FTP Server can be stopped and restarted without disrupting other services running in the system. The server allows two modes of shutdown - partial and total. In a partial shutdown, the FTP Server is stopped, but existing clients are allowed to run to completion. In a total shutdown, the FTP Server is stopped and all client connections are terminated.

As with other products in Nucleus XPROT, the Nucleus FTP Server and Client can operate in an IPv6-only environment. When working in an IPv6-only environment, all IPv4 parts of the software are excluded, resulting in a smaller target footprint.

Nucleus Networking 5.2 and later versions provide a mechanism for services to send messages and advanced debugging information to applications. This mechanism is called the Notification Module. The FTP Server uses the notification module to send debugging information during initialization and termination sequences. The application that requested the FTP Server to start or stop can query the debug queue to determine if any errors occurred while the server was started or stopped.

## FTP Commands

[Table 10-1](#) describes the commands supported with this version of the Nucleus XPROT. If you wish to add more commands, you can modify the `Control_Task` routine found in *fst.c* to add a

call for the command, and write your command in *fsp.c* (for server commands) or create the appropriate *fcp\_command.c* (for client commands).

**Table 10-1. FTP Commands**

Command	Supported	Meaning
abor	Server	This command tells the server to abort the previous ftp service command and any associated transfer of data.
acct	Client/Server	Account command.
allo	Unsupported	This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred.
appe	Client/Server	This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, the data is appended to that file; otherwise, the file specified in the pathname is created at the server site.
cdup	Client	Change to parent directory.
cwd	Client/Server	This command allows you to work with a different directory or dataset for file storage or retrieval without altering login or accounting information.
dele	Client/Server	This command causes the file specified in the pathname to be deleted.
eprt	Client/Server	The argument is a family-host-port specification for the type of communication and data port to be used in the data connection. This command is used over IPv4 and IPv6 connections.
epsv	Client/Server	This command requests the server to “listen” on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. This command is used over IPv4 and IPv6 connections.
feat	Client/Server	This command causes the server to send a list of extended features supported by the server.
help	Client/Server	This command causes the server to send helpful information regarding its implementation status over the control connection to you.
list	Client/Server	This command causes a list of directories and filenames to be sent from the server.
mkd	Client/Server	Make directory command.
mode	Client/Server	Transfer Mode command. Only the MODE = stream command is supported.
nlst	Client/Server	This command causes a list of filenames to be sent.

**Table 10-1. FTP Commands (cont.)**

<b>Command</b>	<b>Supported</b>	<b>Meaning</b>
noop	Client/Server	This command does not affect any parameters or previously entered commands.
pass	Client/Server	Command that specifies the user's password.
pasv	Client/Server	This command requests the server to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. This command is used only over IPv4 connections.
port	Client/Server	The argument is a host-port specification for the data port to be used in data connection. This command is used only over IPv4 connections.
pwd	Client/Server	Print current working directory command.
quit	Client/Server	Logout command.
rein	Unsupported	This command terminates a user, flushing all I/O and account information, except to allow any transfer in progress to be completed.
rest	Client/Server	The argument field of the command represents the server marker at which file transfer is to be restarted.
retr	Client/Server	The retrieve command.
rmd	Client/Server	Remove directory command.
rnfr	Client/Server	This command specifies the old pathname of the file that is to be renamed. This command must be immediately followed by a "rename to" command specifying new file pathname.
rnto	Client/Server	This command specifies the new pathname of the file that is to be renamed. This command must immediately be preceded by a "rename from" command specifying the old file pathname.
site	Unsupported	This command provides site parameters.
size	Client/Server	The argument field of this command contains a path to a file in the server. The server would in turn respond to this command by sending the size of the file whose path was supplied.
smnt	Unsupported	Structure Mount command.
stat	Client/Server	This command causes a status report to be sent over the control connection in the form of a reply.
stor	Client/Server	Store command is used for storing files.
stou	Unsupported	Store unique command.
stru	Client/Server	File structure command.

**Table 10-1. FTP Commands (cont.)**

Command	Supported	Meaning
syst	Client/Server	System command.
type	Client/Server	The argument specifies the representation type command. Only the image parameter is supported at this time.
user	Client/Server	Command that identifies the user.
xmkd	Client/Server	Experimental make directory command.
xpwd	Client/Server	Experimental print current working directory command.
xrmd	Client/Server	Experimental remove directory command.

## FTP Common Defines and Data Structures

This section covers data structures and defines that are used with both the Nucleus FTP Server and Client.

### FTP Defines

[Table 10-2](#) shows return codes used within the Nucleus FTP Server and Client:

**Table 10-2. FTP Server and Client Return Codes**

Define Variable	Value
FTP_CMD_RECEIVED	1
FTP_EOF	0
NU_SUCCESS	0
FTP_INVALID_CLIENT	-1501
FTP_INVALID_TASK	-1502
FTP_STACK_ERROR	-1503
FTP_INVALID_USER	-1504
FTP_INVALID_PASSWORD	-1505
FTP_INVALID_TYPE_CODE	-1506
FTP_BUFFER_OVERRUN	-1507
FTP_BAD_HOST	-1508
FTP_BAD_RESPONSE	-1509
FTP_TIMEOUT	-1510
FTP_NEED_PASSWORD	-1511

**Table 10-2. FTP Server and Client Return Codes (cont.)**

<b>Define Variable</b>	<b>Value</b>
FTP_NEED_ACCOUNT	-1512
FTP_INVALID_ACCOUNT	-1513
FTP_INVALID_STRU_CODE	-1514
FTP_INVALID_MODE_CODE	-1515
FTP_INVALID_BUFFER	-1516
FTP_TRANSFER_ABORT	-1517
FTP_FILE_ERROR	-1518
FTP_BAD_MSG_FORMAT	-1519
FTP_REPLY_BUFFER_OVERRUN	-1520
FTP_INVALID_IP_ADDR	-1521
FTP_BAD_CMD_FORMAT	-1522
FTP_BAD_FILE_DESCRIPTOR	-1523
FTP_WRITE_FAILED	-1524
FTP_FILE_NOT_FOUND	-1525
FTP_NO_FILE_DESCRIPTOR_AVAIL	-1526
FTP_FILE_ALREADY_EXISTS	-1527
FTP_SPECIAL_ACCESS_ATTEMPTED	-1528
FTP_INVALID_FILE_POINTER	-1529
FTP_UNKNOWN_FILE_ERROR	-1530
FTP_SYNTAX_ERROR	-1531
FTP_REGISTRATION_FAILURE	-1532
FTP_OPEN_DRIVE_FAILURE	-1533
FTP_CURRENT_DIR_FAILURE	-1534
FTP_MEMORY	-1535
FTP_SERVICE_UNAVAILABLE	-1536
FTP_CMD_NOT_IMPLEMENTED	-1537
FTP_BAD_CMD_SEQUENCE	-1538
FTP_FILE_UNAVAILABLE	-1539
FTP_INVALID_FILE_NAME	-1540
FTP_CMD_UNRECOGNIZED	-1541

**Table 10-2. FTP Server and Client Return Codes (cont.)**

Define Variable	Value
FTP_UNKNOWN_NETWORK_PROTOCOL	-1542
FTP_INVALID_PARM	-1543
FTPS_CTASK_STRU_MALLOC_FAILURE	-1544
FTPS_CTASK_MALLOC_FAILURE	-1545
FTPS_CTASK_CREATE_FAILURE	-1546
FTPS_CTASK_KERNEL_BIND_FAILURE	-1547
FTPS_SERVER_TERMINATED	-1548

The following are internal constants used by the Nucleus FTP Server and Client:

**Table 10-3. FTP Server and Client Internal Constants**

Define Variable	Value
FTP_WELLKNOWN	21
FTP_VALID_PATTERN	0x10229700L
FILENAME_SIZE	12
FTP_TYPE_IMAGE	0
FTP_TYPE_ASCII	1
FTP_TYPE_LOCAL8	0

## FTP Data Structures

This section covers the structures used within both the Nucleus FTP Server and Client. The [FTP Server-Level Data Structures](#) section describes data structures only used by the FTP server APIs. The [FTP Client-Level Data Structures](#) section describes data structures only used by the FTP client APIs.

### IP\_ADDR

IP\_ADDR is the typedef name for the IP\_ADDR\_STRUCT data structure. IP\_ADDR is used to store the Client's IP address and the port number when a data connection is made.

```
typedef struct IP_ADDR_STRUCT
{
    UINT8    ip_num[MAX_ADDRESS_SIZE];
    INT16    port_num;
} IP_ADDR;
```

The members of the structure are defined in [Table 10-4](#). Some members of this structure have been omitted as they are used internally.

**Table 10-4. IP\_ADDR**

Member	Description
ip_num	IP address of the host.
port_num	Port number to which the service is bound.

## Related Topics

[FTP\\_CLIENT](#)

[NU\\_FTPC\\_Client\\_Open](#)

[NU\\_FTPC\\_Client\\_Open2](#)

# FTP Server-Level Defines and Data Structures

This section covers defines and data structures used in the Nucleus FTP Server. This section is divided into two parts. The two parts consist of:

- defines used within the Nucleus FTP Server
- structure definitions used within the Nucleus FTP Server

## FTP Server-Level Defines

The following are internal user-configurable constants used by the Nucleus FTP Server. The definitions are found in the *os/include/networking/ftp\_cfg.h* file.

**Table 10-5. FTP Server User-Configurable Constants**

Define Variable	Value
FTPS_COMMAND_BUFFER_SIZE	8
FTPS_REPLY_BUFFER_SIZE	1460
FTPS_GENERIC_BUFF_SIZE	256
FTPS_TRANSFER_BUFF_SIZE	512
FTPS_INACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTPS_DATA_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTPS_DATA_TASK_TIMEOUT	(30 * TICKS_PER_SECOND)
FTP_GET_CURRENT_YEAR	2006



The following is a list of task-related user-configurable variables used by the Nucleus FTP Server. These values are optimal, but if you add more functionality, you may need to increase the `STACK_SIZE` values. The definitions are found in the `os/include/networking/ftp_cfg.h` file.

**Table 10-6. FTP Server Task-Related User-Configurable Variables**

Define Variable	Value
<code>CLEANER_QUEUE_SIZE</code>	10
<code>FTPS_SERVER_DEFAULT_PORT</code>	21
<code>FTPS_SERVER_MAX_PENDING</code>	10
<code>FTPS_MASTER_STACK_SIZE</code>	5500
<code>FTPS_CLEANER_STACK_SIZE</code>	2500
<code>FTPS_CONTROL_STACK_SIZE</code>	5500
<code>FTPS_DATA_STACK_SIZE</code>	4500
<code>FTPS_DATA_DIR_STACK_SIZE</code>	2000
<code>CONTROL_PRIORITY</code>	100
<code>DATA_PRIORITY</code>	<code>CONTROL_PRIORITY + 2</code>
<code>DATA_TIME_SLICE</code>	<code>TICKS_PER_SECOND &gt;&gt; 2</code> /* 1/4 of a second */

## FTP\_GET\_CURRENT\_YEAR

When the client requests Nucleus FTP Server for a directory listing, FTP Server checks to see whether the current year is the same as the year the file was created. If it is, then the server sends the time, otherwise it sends the year as part of the directory listing to the client.

This macro is defined by you. It is set to the current year:

```
#define FTP_GET_CURRENT_YEAR 2006
```

## FTP Server-Level Data Structures

No custom data structures are required specifically for the Nucleus [FTP Server-Level Functions](#).

# FTP Server Simple User Authentication

## Simple User Authentication Defines

The Macros are defined in `os/include/networking/ftp_cfg.h`.

The macro `FTP_USE_ANONYMOUS` can be defined if you want to enable the use of anonymous FTP connections. The password for anonymous users is currently an email address. No checking is implemented other than the form of a standard email address. Anonymous FTP is enabled by default (`NU_TRUE`):

```
#define FTP_USE_ANONYMOUS          NU_TRUE
```

The maximum length of a username and password as defined in the following `FTP_Password_List[]` is defined by the following constants:

```
#define FTP_MAX_ID_LENGTH          32  
#define FTP_MAX_PW_LENGTH         32
```

Note that if the version of Nucleus NET used in conjunction with XPROT is release 5.1 or greater, the Nucleus NET User Management Module is used for password management, and the above macros are not used. Refer to the [NET](#) chapter for more information on configuring the Nucleus NET User Management Module.

## Simple User Authentication Data Structures

### FTPSACCT

FTPSACCT is the typedef name for the FTPSacct data structure. The FTPSACCT data structure holds the password list. The [Simple User Authentication Configurable Variable](#) explains how the structure is used to define user access to a server.

```
typedef struct FTPSacct  
{  
    CHAR id[FTP_MAX_ID_LENGTH];  
    CHAR pw[FTP_MAX_PW_LENGTH];  
} FTPSACCT;
```

Where `FTP_MAX_ID_LENGTH` and `FTP_MAX_PW_LENGTH` are explained further in the [Simple User Authentication Defines](#) section of this chapter.

The members of the structure are defined in [Table 10-7](#).

**Table 10-7. FTPSACCT**

Member	Description
id	User ID.
pw	Password for the User ID.

## Related Topics

[Simple User Authentication Data Structures](#)

## Simple User Authentication Configurable Variable

A sample list of usernames and passwords for simple user authentication is defined in *xprot/ftp/src/ftp\_cfg.c*. Edit this list to add new users to this server.

---

### Note



If the version of Nucleus NET used in conjunction with XPROT is less than release 5.1, no dynamic user administration functions are available. However, if the version of Nucleus NET is 5.1 or greater, the User Management Module can be used by the application layer to manage user names and passwords. Refer to the [NET](#) chapter for more information on the Nucleus NET User Management Module.

---

```
FTPSACCT FTP_Password_List[] =
{
    {"jon", "doe"},
    {"fred", "fish"},
    {"joe", "blow"},
    {'\0', '\0'}
};
```

Where the FTPSACCT datastructure is defined in the [Simple User Authentication Data Structures](#) section of this chapter.

## FTP Server-Level Functions

This section describes all function prototypes used within the FTP Server.

- [NU\\_FTP\\_Server\\_Init](#)
- [NU\\_FTP\\_Server\\_Uninit](#)

## NU\_FTP\_Server\_Init

This function is the application layer function that is called to initialize and start the Nucleus FTP Server.

### Usage

```
INT32 NU_FTP_Server_Init (VOID);
```

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **NU\_INVALID**  
A general-purpose error condition. This generally indicates that a required resource (task, semaphore, and so on) could not be created.

### Description

If IPv6 has been included in Nucleus NET, the server services both incoming IPv4 and IPv6 requests. If IPv6 has been excluded from Nucleus NET, the server services only incoming IPv4 requests. If IPv6 has been included but IPv4 has been excluded from Nucleus NET, the server services only incoming IPv6 requests.

### Example

```
STATUS status;  
  
/* Initialize the FTP Server */  
status = NU_FTP_Server_Init();  
  
if (status == NU_SUCCESS)  
/* The FTP Server was successfully initialized */
```

### Related Topics

[FTP Server-Level Functions](#)

## NU\_FTP\_Server\_Uninit

This function is the application layer function that is called to uninitialize and stop the Nucleus FTP Server. A flag can be passed to this function to define the level of urgency required while stopping the FTP Server.

### Usage

```
STATUS NU_FTP_Server_Uninit (UINT8 urgency);
```

### Arguments

- urgency

Flag indicating server shutdown type.

If the urgency flag is set to `NU_TRUE` (complete shutdown), then all clients connected to the FTP Server are disconnected, the server is shutdown and all resources (memory, semaphores, and so on) are deallocated.

If the urgency flag is set to `NU_FALSE` (partial shutdown), then the server is shutdown, but existing clients are allowed to run to completion. In this situation, the server task and its resources are freed while the resources needed to handle existing clients are maintained.

Irrespective of the type of shutdown, the FTP Server can be restarted. On a restart after a partial shutdown, the server accepts new clients and allows for any existing clients to complete operations. On a restart after complete shutdown, all existing clients are terminated and FTP services are restarted, allowing for the server to accept new clients.

### Return Values

- `NU_SUCCESS`.

Upon successful completion.

- `NU_INVALID`

A general-purpose error condition. This indicates that some functions in the shutdown sequence failed.

### Example

```
STATUS status;

/* Sequence of events:
 * - Terminate FTP Server service, allow clients to run
 * - Allow 10 minutes for clients to complete execution
 * - Terminate FTP Server completely, prepare for server restart
 * - Restart FTP Server.
 */
/* Terminate FTP Server, allow existing clients to complete execution
(Partial shutdown) */
status = NU_FTP_Server_Uninit(NU_FALSE);

if (status != NU_SUCCESS)
{
    printf("FTP Server stop failed.\n");
}
```

```
        /* Exit function */
    }

    /* Allow 10 minutes for clients to finish execution. */
    NU_Sleep(10 * 60 * TICKS_PER_SECOND);

    /* Terminate all existing clients to prepare for server restart. */
    status = NU_FTP_Server_Uninit(NU_TRUE); /* Total shutdown */

    if (status == NU_SUCCESS)
    {
        printf("FTP Server completely shutdown\n");

        /* Wait 1 second for cleanup of services to complete. */
        NU_Sleep(TICKS_PER_SECOND);

        /* Restart FTP Server. */
        status = NU_FTP_Server_Init();

        if (status == NU_SUCCESS)
            printf("FTP Server successfully restarted\n");
    }
}
```

## Related Topics

[FTP Server-Level Functions](#)

# FTP Client-Level Defines and Data Structures

## FTP Client-Level Defines

The Nucleus FTP Client API uses a number of user-configurable constants in the form of #defines. The definitions are located in the *os/include/networking/ftp\_cfg.h* file.

**Table 10-8. FTP Client User-Configurable Constants**

Name	Value
FTPC_MAX_RECV_PACKET	1500
FTPC_REPLY_BUFFER_SIZE	512
FTPC_GENERIC_BUFF_SIZE	256
FTPC_TRANSFER_BUFF_SIZE	1024
FTPC_INACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)
FTPC_CCONN_TIMEOUT	(30 * SCK_Ticks_Per_Second)
FTPC_DATACT_TIMEOUT	(300 * SCK_Ticks_Per_Second)

The timeout value, `FTPC_INACT_TIMEOUT`, is the number of seconds to wait for the expected response from the server, on either the control connection or the data connection, before closing the session. This value can be changed to suit individual systems.

The timeout value, `FTPC_CCONN_TIMEOUT`, is the number of seconds to wait after attempting to close the control connection before terminating the connection and reporting an error. This value can be changed to suit individual systems.

The timeout value, `FTPC_DATACT_TIMEOUT`, is the number of seconds to wait after attempting to send or receive data before the connection is closed and an error is reported. This value can be changed to suit individual systems.

## FTP Client-Level Data Structures

The Nucleus FTP Client API relies on a number of structures in order to maintain reentrancy. These structures contain connection-specific data and are required for every call to a function in the API.

## FTP\_CLIENT

`FTP_CLIENT` is the typedef name for the `FTP_CLIENT_STRUCT` data structure and defines the parameters of a specific FTP Client connection. The [NU\\_FTPC\\_Client\\_Open2](#) function initializes the fields in this structure and creates the TCP/IP connection.

```
typedef struct FTP_CLIENT_STRUCT
```

```

{
    UNSIGNED    valid_pattern;
    UNSIGNED    task_id;
    IP_ADDR     *host_addr,
               *local_data_addr;

    INT32       restart;
    INT16       ftpc_family;
    BOOLEAN     mode;
    INT         socketd;
    INT         transfer_type;
    INT         last_error;
    INT         stack_error;
    INT         reply_idx,
               reply_tail;
    CHAR        reply_buff[FTPC_REPLY_BUFFER_SIZE];
} FTP_CLIENT

```

The members of the structure are defined in [Table 10-9](#). Some members of the structure have been omitted as they are used internally.

**Table 10-9. FTP\_CLIENT\_STRUCT**

Member	Description
valid_pattern	A constant value used internally to keep track of the validity of the connection. The <a href="#">NU_FTPC_Client_Open2</a> routine places a valid pattern in the valid_pattern field, which is checked by every other function in the API, and stores the task ID of the calling task in the task_id field.
task_id	Used internally to hold the current task pointer. Every open connection is associated with the task that calls <a href="#">NU_FTPC_Client_Open2</a> and may not be used by any other task.
host_addr	Pointer to internally-used IP address of the remote side.
local_data_addr	Pointer to the client's address. Can be used by the application to fill in the client's address. The local_data_addr field must be initialized by the application layer before calling the GET, PUT, APPEND, DIR and NLIST routines. This field is used in PORT and EPRT calls.
restart	Used internally to store the restart point for the connection.
ftpc_family	Can be used by the application to fill in the client's address family. This field determines the communication type to use for the call; IPv4 or IPv6. This field must contain either NU_FAMILY_IP to specify IPv4 or NU_FAMILY_IP6 to specify IPv6 for the call. The application layer must initialize the ftpc_family field before calling the <a href="#">NU_FTPC_Client_Open2</a> routine.



**Table 10-9. FTP\_CLIENT\_STRUCT (cont.)**

Member	Description
mode	Used internally to determine if the client is connected in an ACTIVE or PASSIVE mode.
socketd	Used internally to hold the client's socket identifier. Once a connection has been established, the socket descriptor for the connection is stored in the socketd field.
transfer_type	Used internally to hold the type of data being transferred. Nucleus currently supports the following types of data: <ul style="list-style-type: none"><li>• ASCII: FTP_TYPE_ASCII</li><li>• IMAGE: FTP_TYPE_IMAGE</li></ul>
last_error	Contains the FTP specific error code.
stack_error	Contains the Nucleus NET specific error code returned by the stack in case of an FTP_STACK_ERROR.
reply_idx	Used internally to parse the messages received from the remote side.
reply_tail	Used internally to denote the end of a specific message received from the remote side.
reply_buff	A buffer that contains the reply received from the remote side. The reply buffer is used to store and construct reply messages from the server.

## Related Topics

[FTP Client-Level Functions](#)[FTP Client-Level Command Primitives](#)[IP\\_ADDR](#)

## FTP Client-Level Functions

This section describes all function prototypes used within the FTP Client.

- [NU\\_FTPC\\_Client\\_Append\\_To\\_File](#)
- [NU\\_FTPC\\_Client\\_ChDir](#)
- [NU\\_FTPC\\_Client\\_Close](#)
- [NU\\_FTPC\\_Client\\_Dir](#)
- [NU\\_FTPC\\_Client\\_Get](#)
- [NU\\_FTPC\\_Client\\_Login](#)

- [NU\\_FTPC\\_Client\\_MkDir](#)
- [NU\\_FTPC\\_Client\\_Nlist](#)
- [NU\\_FTPC\\_Client\\_Open](#)
- [NU\\_FTPC\\_Client\\_Open2](#)
- [NU\\_FTPC\\_Client\\_Put](#)
- [NU\\_FTPC\\_Client\\_Rename\\_File](#)
- [NU\\_FTPC\\_Client\\_Restart](#)
- [NU\\_FTPC\\_Client\\_Rmdir](#)
- [NU\\_FTPC\\_Client\\_Size](#)
- [NU\\_FTPC\\_Client\\_Status](#)
- [NU\\_FTPC\\_Client-Tran\\_Mode](#)
- [NU\\_FTPC\\_Client-Tran\\_Type](#)

## NU\_FTPC\_Client\_Append\_To\_File

This function appends a local file specified by lpath to the remote file at the location specified by rpath.

### Usage

```
INT NU_FTPC_Client_Append_To_File (FTP_CLIENT *client,  
                                   CHAR          *rpath,  
                                   CHAR          *lpath);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- rpath  
Pointer to a buffer containing the path of the remote file.
- lpath  
Pointer to a buffer containing the path of the local file.

### Return Values

- NU\_SUCCESS.  
Upon successful completion
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from the FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize a command.
- FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
- FTP\_INVALID\_USER  
The FTP Server did not recognize the user account.
- FTP\_BAD\_RESPONSE  
The FTP Server returned an unknown error code.

- **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_MEMORY**  
A memory allocation error occurred.
- **FTP\_FILE\_ERROR**  
An internal file i/o error occurred.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        localfile[256], remotefile[256];

/* Connect and log in to FTP Server. */
. . .

/* Populate localfile and remotefile variables with file names. */
strcpy(localfile, "myfile.txt");
strcpy(remotefile, "target.txt");

/* Send the Append command. */
status = NU_FTPC_Client_Append_To_File(&ftp_client, localfile,
                                       remotefile);

if (status == NU_SUCCESS)
    /* Append to file was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_ChDir

NU\_FTPC\_Client\_ChDir attempts to change the currently active directory on the remote system.

### Usage

```
INT NU_FTPC_Client_ChDir (FTP_CLIENT *client,  
                           CHAR        *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to a buffer containing the remote path.

### Return Values

- **NU\_SUCCESS.**  
Upon successful completion.
- **FTP\_INVALID\_CLIENT**  
The client did not receive a “220 Service Ready” message from FTP Server.
- **FTP\_INVALID\_TASK**  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_FILE\_NOT\_FOUND**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.

- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log in to FTP Server. */
. . .

/* Populate path with the new target directory on server. */
strcpy(path, "subdir1");

/* Send the ChDir command. */
status = NU_FTPC_Client_ChDir(&ftp_client, path);

if (status == NU_SUCCESS)
    /* The directory change was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Close

This function closes an FTP connection between the client and the target host. If an error is returned by this function, the connection is still closed. The error indicates that some internal function failed in its task.

### Usage

```
INT NU_FTPC_Client_Close (FTP_CLIENT *client);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
- FTP\_BAD\_RESPONSE  
The FTP Server returned an unknown error code.
- FTP\_INVALID\_PARM  
A required parameter is null.

### Example

```
STATUS      status;  
FTP_CLIENT  ftp_client;  
  
/* Connect and log into the FTP Server */  
. . .  
  
/* Send commands to the server */  
. . .  
  
/* Send the Close Connection command. */  
status = NU_FTPC_Client_Close(&ftp_client);
```

```
if (status == NU_SUCCESS)
    /* The close connection was successful. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



---

## NU\_FTPC\_Client\_Dir

This function performs a directory listing of the current active remote directory, including only those files that match the specification stored in filespec.

### Usage

```
INT NU_FTPC_Client_Dir (FTP_CLIENT *client,  
                        CHAR          *buffer,  
                        INT           buff_size,  
                        CHAR          *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
Pointer to a buffer that holds the directory information.
- **buff\_size**  
Size of the buffer.
- **filespec**  
Pointer to a buffer containing the specification of the files to be listed.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_CLIENT**  
The client did not receive a “220 Service Ready” message from the FTP Server.
- **FTP\_INVALID\_TASK**  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_INVALID\_BUFFER**  
The data buffer is invalid.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

The listing is stored in buffer. If the listing exceeds the amount of space available in buffer (as defined by buff\_size), the function discards the excess input. You can then choose to use the incomplete data or call the function again with a larger buffer.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        buffer[256], filespec[256];
INT         buff_size;

/* Connect and log into the FTP Server */
. . .

/* Populate filespec, clear out buffer and buff_size. */
strcpy(filespec, "subdir1");
strcpy(buffer, "");
buff_size = 0;

/* Send the Dir command. */
status = NU_FTPC_Client_Dir(&ftp_client, buffer, buff_size,
                           filespec);

if (status == NU_SUCCESS)
    /* Directory list was successful, and the result is in buffer. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Get

This function retrieves a file from the host computer specified in `rpath` and stores the file in the local location specified in `lpath`. This function can also restart/resume an interrupted retrieval if REStart markers are set using [NU\\_FTPC\\_Client\\_Restart](#) before calling this function.

### Usage

```
INT NU_FTPC_Client_Get (FTP_CLIENT *client,  
                        CHAR        *rpath,  
                        CHAR        *lpath);
```

### Arguments

- `client`  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- `rpath`  
Pointer to a buffer containing the path of the remote file.
- `lpath`  
Pointer to a buffer containing the path of the local file.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `FTP_INVALID_CLIENT`  
The client did not receive a “220 Service Ready” message from FTP Server.
- `FTP_INVALID_TASK`  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- `FTP_SERVICE_UNAVAILABLE`  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
- `FTP_CMD_UNRECOGNIZED`  
The FTP Server did not recognize the command.
- `FTP_BAD_CMD_FORMAT`  
The FTP Server did not recognize the command format.
- `FTP_INVALID_USER`  
The FTP Server did not recognize the user account.
- `FTP_BAD_RESPONSE`  
The FTP Server returned an unknown error code.

- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found, or the user does not have access to the file.
- **FTP\_FILE\_NOT\_FOUND**  
The FTP Server could not find the requested file.
- **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_MEMORY**  
A memory allocation error occurred.
- **FTP\_FILE\_ERROR**  
A file I/O operation caused an error.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        localpath[256], remotepath[256];

/* Connect and log into the FTP Server */
. . .

/* Populate local and remote paths. */
strcpy(localpath, "myfile.txt");
strcpy(remotepath, "subdir1/file.txt");

/* Send the Get command. */
status = NU_FTPC_Client_Get(&ftp_client, remotepath, localpath);

if (status == NU_SUCCESS)
    /* The file transfer was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Login

This function attempts to open the account of a particular user, defined by username, on the currently connected host.

### Usage

```
INT NU_FTPC_Client_Login (FTP_CLIENT *client,  
                           CHAR      *username,  
                           CHAR      *password);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **username**  
Pointer to a buffer containing the user name.
- **password**  
Pointer to a buffer containing the password.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_CLIENT**  
The client did not receive a “220 Service Ready” message from FTP Server.
- **FTP\_INVALID\_TASK**  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.

- **FTP\_BAD\_CMD\_SEQUENCE**  
The FTP Server returned a “Bad command sequence” error.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

The password parameter may be left NULL if the host does not require a password. If the parameter is left NULL and the host does ask for a password, the function returns **FTP\_INVALID\_PASSWORD**. If a password is not required by the host, but is provided by you, it is ignored. If username is NULL, the function attempts to connect as ‘anonymous,’ with no password.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        username[12], password[12];

/* Connect to FTP server. */
. . .

/* Populate username and password strings. */
strcpy(username, "fred");
strcpy(password, "fish");

/* Send the Login command */
status = NU_FTPC_Client_Login(&ftp_client, username, password);

if (status == NU_SUCCESS)
    /* User login was successful. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_MkDir

This function creates a directory with the name stored in path. An error is returned if path refers to a directory that already exists, the path of the new directory is invalid (an absolute path where one of the parent directories does not exist), or if the user does not have the appropriate privilege level.

### Usage

```
INT NU_FTPC_Client_MkDir (FTP_CLIENT *client,  
                          CHAR        *path);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- path  
Pointer to the character string containing the path of the new directory.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
- FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
- FTP\_CMD\_NOT\_IMPLEMENTED  
The FTP Server informed that the requested command has not been implemented.
- FTP\_INVALID\_USER  
The FTP Server did not recognize the user account.

- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the same.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log into the FTP Server */
. . .

/* Populate the remote path variable. */
strcpy(path, "newdir1");

/* Send the Mkdir command. */
status = NU_FTPC_Client_Mkdir(&ftp_client, path);

if (status == NU_SUCCESS)
    /* Directory create was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FTPC\_Client\_Nlist

This function performs a directory listing of the remote host, including only those files that match the specification stored in filespec. The listing is stored in buffer. If the listing exceeds the amount of space available in buffer (as defined by buff\_size) the listing is truncated.

### Usage

```
INT NU_FTPC_Client_Nlist (FTP_CLIENT *client,  
                          CHAR        *buffer,  
                          INT         buff_size,  
                          CHAR        *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
Pointer to a buffer that holds the directory information.
- **buff\_size**  
Size of the buffer.
- **filespec**  
Pointer to a buffer containing the specification of the files to be listed.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_CLIENT**  
The client did not receive a “220 Service Ready” message from FTP Server.
- **FTP\_INVALID\_TASK**  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.

- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_INVALID\_BUFFER**  
The data buffer is invalid.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        buffer[256], filespec[256];
INT         buff_size;

/* Connect and log into the FTP Server. */
. . .

/* Populate filespec, clear buffer */
strcpy(filespec, "subdir1");
strcpy(buffer, "");
buff_size = 0;

/* Send the Directory List command. */
status = NU_FTPC_Client_Nlist(&ftp_client, buffer, buff_size,
                             filespec);

if (status == NU_SUCCESS)
    /* Directory list was successful, and the result is in buffer. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Open

This routine has been replaced by NU\_FTPC\_Client\_Open2. The routine is supported for backward compatibility.

### Usage

```
INT NU_FTPC_Client_Open (FTP_CLIENT *client,  
                        IP_ADDR   *host_ip);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **host\_ip**  
Pointer to the IP address of the remote host.  
The [IP\\_ADDR](#) data structure is defined in the [FTP Common Defines and Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_BAD\_HOST**  
The host response did not contain correct data.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Description

This function initializes the `FTP_CLIENT` instance `client` and attempts to create an IPv4 connection to the host defined by `host_ip`. Note that if the `port_num` field of `host_ip` is zero, then the default port for FTP connections is used, as defined by `FTP_WELLKNOWN`, which defaults to 21. Once a connection has been successfully made through the use of this call, only the task that made the call may use the connection.

#### Note



Do not call this function with a structure that still refers to an open connection, as the old connection will be lost, and the structure will be overwritten with the data for the new connection.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
IP_ADDR     host_addr;
CHAR        host_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate host_addr variable with IP. */
memcpy(&host_addr.ip_num, host_ip, IP_ADDR_LEN);

/* Send Open command */
status = NU_FTPC_Client_Open(&ftp_client, &host_addr);
if (status == NU_SUCCESS)
    /* Connection to the server was successful. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP Common Defines and Data Structures](#)

[FTP\\_CLIENT](#)

[IP\\_ADDR](#)

## NU\_FTPC\_Client\_Open2

This function initializes the FTP\_CLIENT instance client and attempts to create a connection to the host defined by host\_ip using the network protocol specified in the ftpc\_family parameter of client.

### Usage

```
INT NU_FTPC_Client_Open2 (FTP_CLIENT *client,  
                          IP_ADDR    *host_ip);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- host\_ip  
Pointer to the IP address of the remote host.  
The [IP\\_ADDR](#) data structure is defined in the [FTP Common Defines and Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_BAD\_HOST  
The host response did not contain correct data.
- FTP\_INVALID\_PARM  
A required parameter is null.

### Description

Once a connection has been successfully made through the use of this call, only the task that made the call may use the connection.

---

**Note**

Note that if the port\_num field of host\_ip is zero, then the default port for FTP connections is used, as defined by FTP\_WELLKNOWN, which defaults to 21.

---

---

**Note**

Do not call this function with a structure that still refers to an open connection, as the old connection will be lost, and the structure will be overwritten with the data for the new connection.

---

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
IP_ADDR     host_addr;
CHAR        host_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate the host_addr variable with IP. */
memcpy(&host_addr.ip_num, host_ip, IP_ADDR_LEN);

/* Send the Open command. */
status = NU_FTPC_Client_Open2(&ftp_client, &host_addr);

if (status == NU_SUCCESS)
    /* Connection to the server was successful. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP Common Defines and Data Structures](#)

[FTP\\_CLIENT](#)

[IP\\_ADDR](#)

## NU\_FTPC\_Client\_Put

This function transfers a local file specified by lpath to the remote host, storing the file at the location specified by rpath.

### Usage

```
INT NU_FTPC_Client_Put (FTP_CLIENT *client,  
                        CHAR          *rpath,  
                        CHAR          *lpath);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- rpath  
Pointer to a buffer containing the path of the remote file.
- lpath  
Pointer to a buffer containing the path of the local file.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
- FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
- FTP\_INVALID\_USER  
The FTP Server did not recognize the user account.
- FTP\_BAD\_RESPONSE  
The FTP Server returned an unknown error code.

- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to it.
- **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_INVALID\_FILE\_NAME**  
The FTP Server could not find the requested file.
- **FTP\_MEMORY**  
A memory allocation error occurred.
- **FTP\_FILE\_ERROR**  
A file i/o operation caused an error.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

If the file exists on the host, and the host is configured to not allow overwrites, an error is returned. This function can also restart/resume an interrupted file transfer if REStart markers are set using the `NU_FTPC_Client_Restart()` function before calling this function.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        localpath[256], remotepath[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate local and remote paths. */
strcpy(localpath, "myfile.txt");
strcpy(remotepath, "subdir1/file.txt");

/* Send the Put command. */
status = NU_FTPC_Client_Put(&ftp_client, remotepath, localpath);

if (status == NU_SUCCESS)
    /* File transfer to server was successful. */
```



## Related Topics

[FTP Client-Level Functions](#)

[FTP\\_CLIENT](#)

[FTP Client-Level Defines and Data Structures](#)

## NU\_FTPC\_Client\_Rename\_File

This function changes the name of a remote file from `old_file_name` to `new_file_name`. An error is returned if the file does not exist or if a file already exists with the name in `new_file_name`.

### Usage

```
INT NU_FTPC_Client_Rename_File (FTP_CLIENT *client,  
                                CHAR        *old_file_name,  
                                CHAR        *new_file_name);
```

### Arguments

- `client`  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- `old_file_name`  
Pointer to the old file name.
- `new_file_name`  
Pointer to the new file name.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- `FTP_INVALID_CLIENT`  
The client did not receive a “220 Service Ready” message from FTP Server.
- `FTP_INVALID_TASK`  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- `FTP_STACK_ERROR`  
A send or receive command returned an error while communicating with the FTP Server.
- `FTP_SERVICE_UNAVAILABLE`  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- `FTP_CMD_UNRECOGNIZED`  
The FTP Server did not recognize the command.
- `FTP_BAD_CMD_FORMAT`  
The FTP Server did not recognize the command format.
- `FTP_CMD_NOT_IMPLEMENTED`  
The FTP Server informed that the requested command has not been implemented.

- **FTP\_BAD\_CMD\_SEQUENCE**  
The FTP Server returned a “Bad command sequence” error.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received the account information needed to process the requested action.
- **FTP\_INVALID\_FILE\_NAME**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the same.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        old_name[256], new_name[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate old_name and new_name variables */
strcpy(old_name, "oldfile.txt");
strcpy(new_name, "newfile.txt");

/* Send the Rename command. */
status = NU_FTPC_Client_Rename_File(&ftp_client, old_name,
                                     new_name);

if (status == NU_SUCCESS)
    /* File rename was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Restart

This function sets the restart marker for the subsequent file transfer.

### Usage

```
INT FTPC_Client_Restart(FTP_CLIENT *client,  
                        INT32      restartpt);
```

### Arguments

- **client**  
Pointer to a valid FTP client structure  
Data structure [FTP\\_CLIENT](#) is defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **restartpt**  
Integer specifying the restart point

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_PARM**  
A required parameter is null.
- Last error code generated in its internal function calls.

### Description

The function primitive [NU\\_FCP\\_Client\\_REST](#) must be called after this function in order to send the restart marker to the other side. The sole purpose of this function is to set the restart marker which should be sent by calling [NU\\_FCP\\_Client\\_REST](#) subsequently. An appropriate error is returned in case of any failures.

#### Note



If you plan to use [NU\\_FTPC\\_Client\\_Get](#) or [NU\\_FTPC\\_Client\\_Put](#) for get/put functionality, there is no need to invoke the primitive [NU\\_FCP\\_Client\\_REST](#) as explained above because this invocation already happens in these implementations. To resume a successful transfer using [NU\\_FTPC\\_Client\\_Get](#) and [NU\\_FTPC\\_Client\\_Put](#), call [NU\\_FTPC\\_Client\\_Restart](#) before [NU\\_FTPC\\_Client\\_Get](#) or [NU\\_FTPC\\_Client\\_Put](#).

### Example

```
STATUS status;  
FTP_CLIENT ftp_client;  
  
/* Connect and log into the FTP Server. */  
. . .
```

```
/* Set the Restart markers. */  
status = NU_FTPC_Client_Restart(&ftp_client, 532849);  
  
if (status == NU_SUCCESS)  
    /* If successful, the subsequent file transfer  
     * resumes from 532849 bytes. */
```

## Related Topics

[FTP Client-Level Functions](#)

[FTP\\_CLIENT](#)

[NU\\_FTPC\\_Client\\_Restart](#)

[NU\\_FTPC\\_Client\\_Put](#)

[FTP Client-Level Defines and Data Structures](#)

[NU\\_FCP\\_Client\\_REST](#)

[NU\\_FTPC\\_Client\\_Get](#)

## NU\_FTPC\_Client\_RmDir

This function removes the directory specified by path from the server. An error is returned if the directory does not exist, the directory cannot be removed (in use, or not empty), or if you do not have the appropriate privilege level.

### Usage

```
INT NU_FTPC_Client_RmDir (FTP_CLIENT *client,  
                          CHAR        *path);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- path  
Pointer to the character string containing the path of the target directory.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
- FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
- FTP\_CMD\_NOT\_IMPLEMENTED  
The FTP Server informed that the requested command has not been implemented.
- FTP\_BAD\_CMD\_SEQUENCE  
The FTP Server returned a “Bad command sequence” error.

- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to it.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate path with the location of the directory to be deleted. */
strcpy(path, "subdir1");

/* Send the RmDir command. */
status = NU_FTPC_Client_RmDir(&ftp_client, path);

if (status == NU_SUCCESS)
    /* The directory delete was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Size

This function returns the size of a file on the remote host. This function is a client-level wrapper for the `NU_FCP_Client_SIZE()` function. Its sole purpose is to add a caller verification wrapper to the primitive.

### Usage

```
INT  FTPC_Client_Size(FTP_CLIENT *client,
                      CHAR        *path,
                      INT32        *size);
```

### Arguments

- **client**  
Pointer to a valid FTP client structure  
Data structure `FTP_CLIENT` is defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to a character string defining object to be checked
- **size**  
Pointer to the size of the file. On a successful return this contains the size of the requested file.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_PARM**  
A required parameter is null.
- The last error code generated in its internal function calls.

### Example

```
STATUS      status;
INT32       size;
FTP_CLIENT  ftp_client;

/* Connect and log into the FTP Server. */
. . .

/* Send 'size' request. */
status = NU_FTPC_Client_Size(&ftp_client, "ftp_test.txt", &size);

if (status == NU_SUCCESS)
/* If successful, 'size' contains the size the requested file. */
```



## Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Status

This function returns the status of the entity defined by path. The status information is returned in buffer, with a maximum length of bufsize. An error is returned if the entity in path does not exist.

### Usage

```
INT NU_FTPC_Client_Status (FTP_CLIENT *client,  
                           CHAR        *path,  
                           CHAR        *buffer,  
                           INT         bufsize);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- path  
Pointer to the character string containing the object that is checked.
- buffer  
Pointer to a buffer to hold the status information.
- bufsize  
Size of the buffer.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.

- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found, or the user does not have access to the file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256], buffer[256];
INT         buff_size;

/* Connect and log into the FTP Server. */
. . .

/* Populate path and clear buffer. */
strcpy(path, "subdir1");
strcpy(buffer, "");
buff_size = 0;

/* Send the Status command. */
status = NU_FTPC_Client_Status(&ftp_client, path, buffer,
                               buff_size);

if (status == NU_SUCCESS)
    /* Status was successful, and the result is in buffer. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client\_Tran\_Mode

This function alters the defined data transfer mode. The only defined mode is STREAM. The constant FTPC\_MODE\_STREAM defines the value that this function uses to represent this mode.

### Usage

```
INT NU_FTPC_Client_Tran_Mode (FTP_CLIENT *client,  
                              INT          mode);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- mode  
Transfer mode.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
- FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- FTP\_INVALID\_MODE\_CODE  
The mode code parameter is invalid.
- FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
- FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
- FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
- FTP\_CMD\_NOT\_IMPLEMENTED  
The FTP Server informed that the requested command has not been implemented.

- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
INT         mode;

/* Connect and log into the FTP Server. */
. . .

/* Populate mode with mode of transfer desired. */
mode = FTPC_MODE_STREAM;

/* Send the Transfer Mode command. */
status = NU_FTPC_Client-Tran_Mode(&ftp_client, mode);

if (status == NU_SUCCESS)
    /* Transfer mode set was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FTPC\_Client-Tran\_Type

This function alters the defined data transfer type.

### Usage

```
INT NU_FTPC_Client-Tran_Type (FTP_CLIENT *client,  
                              INT          type);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **type**  
The data transfer type. The three possible codes are:
  - FTP\_TYPE\_ASCII
  - FTP\_TYPE\_IMAGE
  - FTP\_TYPE\_LOCAL8 (functionally identical to FTP\_TYPE\_IMAGE).

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_INVALID\_CLIENT**  
The client did not receive a “220 Service Ready” message from FTP Server.
- **FTP\_INVALID\_TASK**  
The task ID of the calling task does not match the task ID in the [FTP\\_CLIENT](#) structure.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_INVALID\_TYPE\_CODE**  
The type code parameter is invalid.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.

- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
INT         type;

/* Connect and log into the FTP Server. */
. . .

/* Populate the mode with the type of transfer desired. */
type = FTP_TYPE_ASCII;

/* Send the Transfer Type command. */
status = NU_FTPC_Client-Tran_Type(&ftp_client, type);

if (status == NU_SUCCESS)
    /* Transfer type set was successful. */
```

### Related Topics

[FTP Client-Level Functions](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## FTP Client-Level Command Primitives

- [NU\\_FCP\\_Client\\_ACCT](#)
- [NU\\_FCP\\_Client\\_APPE](#)
- [NU\\_FCP\\_Client\\_CDUP](#) [NU\\_FCP\\_Client\\_XCUP](#)
- [NU\\_FCP\\_Client\\_CWD](#) [NU\\_FCP\\_Client\\_XCWD](#)
- [NU\\_FCP\\_Client\\_DELE](#)
- [NU\\_FCP\\_Client\\_EPRT](#)
- [NU\\_FCP\\_Client\\_EPSV](#)

- NU\_FCP\_Client\_FEAT
- NU\_FCP\_Client\_HELP
- NU\_FCP\_Client\_LIST
- NU\_FCP\_Client\_MKD NU\_FCP\_Client\_XMKD
- NU\_FCP\_Client\_MODE
- NU\_FCP\_Client\_NLST
- NU\_FCP\_Client\_NOOP
- NU\_FCP\_Client\_PASS
- NU\_FCP\_Client\_PASV
- NU\_FCP\_Client\_PORT
- NU\_FCP\_Client\_PWD NU\_FCP\_Client\_XPWD
- NU\_FCP\_Client\_QUIT
- NU\_FCP\_Client\_REST
- NU\_FCP\_Client\_RETR
- NU\_FCP\_Client\_RMD NU\_FCP\_Client\_XRMD
- NU\_FCP\_Client\_RNFR
- NU\_FCP\_Client\_RNTO
- NU\_FCP\_Client\_STAT
- NU\_FCP\_Client\_STOR
- NU\_FCP\_Client\_STRU
- NU\_FCP\_Client\_SYST
- NU\_FCP\_Client\_Trans\_Ack
- NU\_FCP\_Client\_TYPE
- NU\_FCP\_Client\_USER
- NU\_FCP\_Client\_Verify\_Caller
- NU\_FCP\_Reply\_Read



## NU\_FCP\_Client\_ACCT

This function provides a primitive to send an FTP ACCT message to a server. This function (most likely) returns an error, if it is not directly preceded by either a user or a pass command. The error is returned based on the implementation of the server.

### Usage

```
INT NU_FCP_Client_ACCT (FTP_CLIENT *client,  
                        CHAR         *account);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **account**  
Pointer to the string containing the account name.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_INVALID\_ACCOUNT**  
The account pointer should not be null.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_BAD\_CMD\_SEQUENCE**  
The FTP Server returned a “Bad command sequence” error.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        account[256], username[12], password[12];

/* Connect and log into the FTP Server. */
. . .

/* Populate account with the desired values. */
strcpy(account, "johndoe");

/* Send the Account command. */
status = NU_FCP_Client_ACCT(&ftp_client, account);

if (status == NU_SUCCESS)
    /* Account information transfer was successful. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_APPE

This function provides a primitive to send an FTP APPE message to the server. Any data sent over the data connection is added to a file of that name on the host system. If a path is included in the string, it refers to the file on the host rather than the file to be sent. Wildcard values are not allowed.

### Usage

```
INT NU_FCP_Client_APPE (FTP_CLIENT *client,  
                        CHAR *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **filespec**  
Pointer to the string containing the name of the appended file.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found, or the user does not have access to the file.
- **FTP\_MEMORY**  
A memory allocation error occurred.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_INVALID\_FILE\_NAME**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        filespec[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate filespec with the desired file name. */
strcpy(filespec, "myfile.txt");

/* Send the Append command. */
status = NU_FCP_Client_APPE(&ftp_client, filespec);

if (status == NU_SUCCESS)
    /* File append was successful. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_CDUP NU\_FCP\_Client\_XCUP

These functions provide primitives to send an FTP CDUP (change directory to parent) message to the server.

### Usage

```
INT NU_FCP_Client_CDUP (FTP_CLIENT *client);
```

```
INT NU_FCP_Client_XCUP (FTP_CLIENT *client);
```

### Arguments

- client

Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS

Upon successful completion.

Otherwise, the routines return an operating system-specific error code, Nucleus NET error code or one of the following Nucleus XPROT error codes:

- FTP\_STACK\_ERROR

A send or receive command returned an error while communicating with the FTP Server.

- FTP\_SERVICE\_UNAVAILABLE

The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.

- FTP\_CMD\_UNRECOGNIZED

The FTP Server did not recognize the command.

- FTP\_BAD\_CMD\_FORMAT

The FTP Server did not recognize the command format.

- FTP\_CMD\_NOT\_IMPLEMENTED

The FTP Server informed that the requested command has not been implemented.

- FTP\_INVALID\_USER

The FTP Server did not recognize the user account.

- FTP\_FILE\_NOT\_FOUND

The FTP Server could not find the requested file.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

These functions cause the server to change the working directory to the parent of the current directory. An error from the server indicates that the path was invalid or was inaccessible (due to privilege level). Note that the XCUP version is for backward compatibility with older servers that may not recognize the newer forms. The CDUP variant will call the XCUP variant if an unrecognized command error is received. The XCUP function need not ever be used, and is here for reference only.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;

/* Connect and login to the FTP Server */
. . .

/* Perform the required directory change actions. */
. . .

/* Send the Directory UP command. */
status = NU_FCP_Client_CDUP(&ftp_client);

if (status == NU_SUCCESS)
    /* Directory change was successful. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_CWD NU\_FCP\_Client\_XCWD

These functions provide primitives to send an FTP CWD (change working directory) message to the server.

### Usage

```
INT NU_FCP_Client_CWD (FTP_CLIENT *client,  
                      CHAR *path);  
  
INT NU_FCP_Client_XCWD (FTP_CLIENT *client,  
                      CHAR *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the new directory.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
  
Otherwise, the routines return an operating system-specific error code, Nucleus NET error code, or one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
  - **FTP\_INVALID\_USER**

The FTP Server did not recognize the user account.

- FTP\_FILE\_NOT\_FOUND

The FTP Server could not find the requested file.

- FTP\_BAD\_RESPONSE

The FTP Server returned an unknown error code.

- FTP\_INVALID\_PARM

A required parameter is null.

## Description

An error from the server indicates that the path was either not valid or was inaccessible (privileged access areas, and so on). Note that the XCWD version is for backward compatibility with older servers that may not recognize the newer forms. The CWD variant calls the XCWD variant if an unrecognized command error is received. The XCWD function need not ever be used, and is here for reference only.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate path with the target directory to which to change. */
strcpy(path, "subdir1");

/* Send the Directory Change command. */
status = NU_FCP_Client_CWD(&ftp_client, path);

if (status == NU_SUCCESS)
    /* Directory change was successful. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_DELE

This function provides a primitive to send an FTP DELE message to the server.

### Usage

```
INT NU_FCP_Client_DELE (FTP_CLIENT *client,  
                        CHAR *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the file to delete.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found, or the user does not have access to the file.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

The path parameter is a pointer to a character string containing the name of a remote file to delete. If a path is included in the string, it refers to a path on the server machine. An error indicates that the file did not exist or that the file cannot be deleted.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log into the FTP Server. */
. . .

/* Populate path with filename to delete */
strcpy(path, "deleteme.txt");

/* Send the File Delete command. */
status = NU_FCP_Client_DELE(&ftp_client, path);

if (status == NU_SUCCESS)
    /* File delete was successful. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_EPRT

This function provides a primitive to send an FTP EPRT message to the server. The family type specified in the EPRT message is based on the family type specified in the servaddr parameter.

### Usage

```
INT NU_FCP_Client_EPRT (FTP_CLIENT      *client,  
                        struct addr_struct *servaddr);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **servaddr**  
Pointer to the structure containing the address sent to the server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;  
FTP_CLIENT  ftp_client;  
struct addr_struct server_addr;
```

```
CHAR server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate server_addr with FTP Server's address. */
server_addr.family = NU_FAMILY_IP;

/* Get an unique port number. */
server_addr.port    = 2000 ;

/* Copy the IP address of server to server addr struct. */
memcpy(server_addr.id.is_ip_addrs, server_ip, IP_ADDR_LEN);

/* Connect and log into the FTP Server. */
. . .

/* Send the Extended Data Port command. */
status = NU_FCP_Client_EPRT(&ftp_client, &server_addr);

if (status == NU_SUCCESS)
    /* Extended data port command was successful. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_EPSV

This function provides a primitive to send an FTP EPSV (extended passive connect) message to the server. An error indicates that the server does not recognize or allow this command.

### Usage

```
INT NU_FCP_Client_EPSV (FTP_CLIENT      *client,  
                        struct addr_struct *servaddr);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **servaddr**  
Pointer to the structure containing the address to retrieve from the server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
  
Otherwise, the routine returns an operating system specific error code, Nucleus NET error code or one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
  - **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Example

```
STATUS          status;
FTP_CLIENT      ftp_client;
struct addr_struct server_addr;
CHAR server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate server_addr with the FTP Server's address. */
server_addr.family = NU_FAMILY_IP;

/* Get an unique port number. */
server_addr.port = 2000 ;

/* Copy the IP address of server to the server addr struct. */
memcpy(server_addr.id.is_ip_addrs, server_ip, IP_ADDR_LEN);

/* Connect and log into the FTP Server. */
. . .

/* Send the Extended Passive command. */
status = NU_FCP_Client_EPSV(&ftp_client, &server_addr);

if (status == NU_SUCCESS)
    /* Extended passive command was successful. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_FEAT

This function provides a primitive to send an FTP FEAT message to the server. Parameters include a pointer to a valid FTP client structure, a pointer to a character buffer to store the feat response, and the size of the buffer. Parsing and utilizing the FEAT response (as necessary) from the 'buffer' is at the discretion of the user's application.

### Usage

```
INT FCP_Client_FEAT(FTP_CLIENT *client,  
                   CHAR *buffer,  
                   INT buffsize);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
Pointer to storage location for feat response.
- **buffsize**  
Size of the buffer.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that the requested command has not been implemented.
  - **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
  - **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS      status;
CHAR        *feat_buffer;
FTP_CLIENT  ftp_client;

/* Connect and log into the FTP Server */
. . .

/* Allocate memory for the FEAT-response buffer. */
. . .

/* Send 'FEAT' request. */
status = NU_FCP_Client_FEAT(&ftpc_client, feat_buffer, 2048);

if (status == NU_SUCCESS)
/* If successful, 'buffer' contains the response from the server.
 * This response should then be parsed for an appropriate action
 * based on the available extended features in the server.
 */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_HELP

This function provides a primitive to send an FTP HELP message to the server. The topic parameter is a pointer to a character string containing the topic for which help is requested. An empty string indicates a request for general help.

### Usage

```
INT NU_FCP_Client_HELP(FTP_CLIENT *client,  
                       CHAR        *topic,  
                       CHAR        *buffer,  
                       INT         bufsize);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **topic**  
Pointer to the character string containing the help topic.
- **buffer**  
Pointer to the storage location for help information.
- **bufsize**  
Size of the buffer.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**

The FTP Server informed that the requested command has not been implemented.

- **FTP\_FILE\_UNAVAILABLE**

The file on the FTP Server was either not found or the user does not have access to the same.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        topic[12], buffer[256];
INT         buff_size;

/* Connect and log into the FTP Server. */
. . .

/* Populate the topic with the command for which help is needed. */
strcpy(topic, "LIST");
strcpy(buffer, "");
buff_size = 0;

/* Send the Help command. */
status = NU_FCP_Client_HELP(&ftp_client, topic, buffer,
                           buff_size);

if (status == NU_SUCCESS)
    /* Help request was successful, and the result is in buffer. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_LIST

This function provides a primitive to send an FTP LIST message to the server.

### Usage

```
INT NU_FCP_Client_LIST(FTP_CLIENT *client,  
                      CHAR *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **filespec**  
Pointer to the string containing the directory to be retrieved.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_INVALID\_BUFFER**  
The data buffer is invalid.
  - **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;  
FTP_CLIENT  ftp_client;  
CHAR        filespec[256];  
  
/* Connect and log into the FTP Server */  
. . .  
  
/* Populate filespec with the appropriate values. */  
strcpy(filespec, "subdir1");  
  
/* Send the List command. */  
status = NU_FCP_Client_LIST(&ftp_client, filespec);
```

```
if (status == NU_SUCCESS)
    /* List request was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_MKD NU\_FCP\_Client\_XMKD

These functions provide primitives to send an FTP MKD (make directory) message to the server.

### Usage

```
INT NU_FCP_Client_MKD(FTP_CLIENT *client,  
                      CHAR *path);  
  
INT NU_FCP_Client_XMKD(FTP_CLIENT *client,  
                       CHAR *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the new directory.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routines return one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server informed that the requested command has not been implemented.
  - **FTP\_INVALID\_USER**

The FTP Server did not recognize the user account.

- **FTP\_FILE\_UNAVAILABLE**

The file on the FTP Server was either not found or the user does not have access to the file.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Description

An error from the server indicates that the path was invalid or that creating a directory was a privileged action. Note that the XMKD version is for backward compatibility with older servers that may not recognize the newer forms. The MKD variant calls the XMKD variant if an unrecognized command error is received. The XMKD function need not ever be used, and is here for reference only.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path[256];

/* Connect and log into the FTP Server */
. . .

/* Populate path with the appropriate values. */
strcpy(path, "subdir1");

/* Send the MkDir command. */
status = NU_FCP_Client_MKD(&ftp_client, path);

if (status == NU_SUCCESS)
    /* Path was successfully created. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_MODE

This function provides a primitive to send an FTP MODE message to the server. The only supported mode is STREAM.

### Usage

```
INT NU_FCP_Client_MODE(FTP_CLIENT *client,  
                       INT mode);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **mode**  
Code for transfer mode.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
  - **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
INT         mode;

/* Connect and log into the FTP Server. */
. . .

/* Populate Mode with the desired transfer mode. */
mode = FTTPC_MODE_STREAM;

/* Send the List command */
status = NU_FCP_Client_MODE(&ftp_client, mode);

if (status == NU_SUCCESS)
    /* Mode command was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_NLST

This function provides a primitive to send an FTP NLST message to the server.

### Usage

```
INT NU_FCP_Client_NLST(FTP_CLIENT *client,  
                       CHAR *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **filespec**  
Pointer to the string containing the directory to be retrieved.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the same.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that the requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;  
FTP_CLIENT  ftp_client;  
CHAR        filespec[256];  
  
/* Connect and log into the FTP Server. */  
. . .  
  
/* Populate filespec with the appropriate values. */  
strcpy(filespec, "subdir1");  
  
/* Send the Machine-readable List command. */  
status = NU_FCP_Client_NLST(&ftp_client, filespec);  
  
if (status == NU_SUCCESS)  
    /* Machine-readable list request was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_NOOP

This function provides a primitive to send an FTP NOOP (no operation) message to the server. It is only included for RFC-compliance, as it does not do anything of substance.

### Usage

```
INT NU_FCP_Client_NOOP(FTP_CLIENT *client);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
  - **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;

/* Connect and log into the FTP Server. */
. . .

/* Send the No Operation command. */
status = NU_FCP_Client_NOOP(&ftp_client);

if (status == NU_SUCCESS)
```

```
/* No operation command was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_PASS

This function provides a primitive to send an FTP PASS message to the server.

### Usage

```
INT NU_FCP_Client_PASS (FTP_CLIENT *client,  
                        CHAR *password);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **password**  
Pointer to the string containing the password.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
- **FTP\_BAD\_CMD\_SEQUENCE**  
The FTP Server returned a “Bad command sequence” error.
- **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

This function has multiple success conditions, including the generic ‘success’ message, indicating that no further action is necessary, and the ‘need account’ message, indicating that an account name (transferred via the ACCT function) is necessary. Note that this function (most likely) returns an error if it is not preceded by a user command. The error is returned based on the implementation of the server.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        username[12], password[12];

/* Connect to FTP Server. */
. . .

/* Populate the username and password values. */
strcpy(username, "fred");
strcpy(password, "fish");

/* Send the USER command to FTP Server. */
status = NU_FCP_Client_USER(&ftp_client, username);

if (status != NU_SUCCESS)
    /* Error sending username, so quit application. */

/* Send the PASS command to FTP Server. */
status = NU_FCP_Client_PASS(&ftp_client, password);

if (status == NU_SUCCESS)
    /* Password was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_PASV

This function provides a primitive to send an FTP PASV (passive connect) message to the server. An error indicates that the server does not recognize or allow this command.

### Usage

```
INT NU_FCP_Client_PASV(FTP_CLIENT *client,  
                      struct addr_struct *servaddr);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **servaddr**  
Pointer to the structure containing the address to retrieve from the server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that the requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
  - **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS          status;
FTP_CLIENT      ftp_client;
struct addr_struct server_addr;
CHAR server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate the server_addr family with the target address family. */
server_addr.family = NU_FAMILY_IP;

/* Get an unique port number. */
server_addr.port   = 2000 ;

/* Copy the IP address of server to server addr struct. */
memcpy(server_addr.id.is_ip_addrs, server_ip, IP_ADDR_LEN);

/* Connect and log into the FTP Server. */
. . .

/* Send the Passive command. */
status = NU_FCP_Client_PASV(&ftp_client, &server_addr);

if (status == NU_SUCCESS)
    /* Passive command was successful. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_PORT

This function provides a primitive to send an FTP PORT message to the server.

### Usage

```
INT NU_FCP_Client_PORT(FTP_CLIENT      *client,  
                      struct addr_struct *servaddr);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **servaddr**  
Pointer to the structure containing the address to be sent to the server.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
  - **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
  - **FTP\_INVALID\_PARM**  
A required parameter is null.

## Example

```
STATUS          status;
FTP_CLIENT      ftp_client;
struct addr_struct server_addr;
CHAR server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Populate server_addr family with target address family. */
server_addr.family = NU_FAMILY_IP;

/* Get an unique port number. */
server_addr.port    = 2000 ;

/* Copy the IP address of server to server addr struct. */
memcpy(server_addr.id.is_ip_addrs, server_ip, IP_ADDR_LEN);

/* Connect and log into the FTP Server. */
. . .

/* Send the PORT command. */
status = NU_FCP_Client_PORT(&ftp_client, &server_addr);

if (status == NU_SUCCESS)
    /* Port command was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_PWD NU\_FCP\_Client\_XPWD

These functions provide primitives to send an FTP PWD (print working directory) message to the server.

### Note



The XPWD version is for backward compatibility with older servers that may not recognize the newer forms. The PWD variant calls the XPWD variant if an unrecognized command error is received. The XPWD function need not ever be used and is here for reference only.

---

### Usage

```
INT NU_FCP_Client_PWD (FTP_CLIENT *client,
                      CHAR *buffer,
                      INT bufsize);

INT NU_FCP_Client_XPWD(FTP_CLIENT *client,
                      CHAR *buffer,
                      INT bufsize);
```

### Arguments

- **client**  
 Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
 Pointer to the storage location of the directory name.
- **bufsize**  
 Size of the buffer.

### Return Values

- **NU\_SUCCESS**  
 Upon successful completion.
- Otherwise, the routines return one of the following Nucleus XPROT error codes:
- **FTP\_STACK\_ERROR**  
 A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
 The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**

The FTP Server did not recognize the command.

- **FTP\_BAD\_CMD\_FORMAT**

The FTP Server did not recognize the command format.

- **FTP\_CMD\_NOT\_IMPLEMENTED**

The FTP Server replied that requested command has not been implemented.

- **FTP\_FILE\_UNAVAILABLE**

The file on the FTP Server was either not found or the user does not have access to the same.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        buffer;
INT         buff_size;

/* Connect and log into the FTP Server */
. . .

/* Clear buffer. */
strcpy(buffer, "");
buff_size = 0;

/* Send the PWD command. */
status = NU_FCP_Client_PWD(&ftp_client, &buffer, buff_size);

if (status == NU_SUCCESS)
    /* PWD successfully sent, the result is in buffer. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_QUIT

This function provides a primitive to send an FTP QUIT message to the server. This function signals to the server that the client desires an end of the connection. The actual TCP close is initiated by the server.

### Usage

```
INT NU_FCP_Client_QUIT (FTP_CLIENT *client);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
- **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;

/* Connect and log into the FTP Server. */
. . .

/* Perform the required actions on the server. */
. . .

/* Send the QUIT command */
status = NU_FCP_Client_QUIT(&ftp_client);

if (status == NU_SUCCESS)
    /* Connection to the server is closed. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_REST

This function provides a primitive to send an FTP REST message to the server. Parameters include a pointer to a valid FTP client structure, and an integer specifying the restart point.

### Usage

```
INT FCP_Client_REST(FTP_CLIENT *client,  
                   INT32      restartpt);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **restartpt**  
Integer specifying the restart point

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a 'Service Unavailable' message. This may be a reply to any command if the ftp service knows it must shut down.
  - **FTP\_FILE\_UNAVAILABLE**  
FTP Server returned a 'File Unavailable' message.
  - **FTP\_MEMORY**  
A memory allocation error occurred.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
FTP Server did not recognize command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**

The FTP Server replied that the requested command has not been implemented.

- **FTP\_INVALID\_USER**

The FTP Server did not recognize the user account.

- **FTP\_NEED\_ACCOUNT**

FTP Server has not received account information that is needed to process the requested action.

- **FTP\_INVALID\_FILE\_NAME**

FTP Server returned an 'Invalid filename' message.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS status;
FTP_CLIENT ftp_client;

/* Connect and log into the FTP Server. */
. . .

/* Send the Restart marker to the remote side. */
status = NU_FCP_Client_REST(&ftp_client, 532849);

if (status == NU_SUCCESS)
/* If successful, the subsequent file transfer
 * resumes from 532849 bytes. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_RETR

This function provides a primitive to send an FTP RETR message to the server. If a path is included in the string, it refers to the path of the source file, not the destination file. Wildcards are not allowed.

### Usage

```
INT NU_FCP_Client_RETR (FTP_CLIENT *client,  
                        CHAR        *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **filespec**  
Pointer to the string containing the name of the file to be retrieved.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the file.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_FILE\_NOT\_FOUND**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        filespec;

/* Connect and log into the FTP Server. */
. . .

/* Populate filespec with the appropriate values. */
strcpy(filespec, "myfile.txt");

/* Send the RETR command. */
status = NU_FCP_Client_RETR(&ftp_client, &filespec);

if (status == NU_SUCCESS)
    /* Retrieve command was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_RMD NU\_FCP\_Client\_XRMD

These functions provide primitives to send an FTP RMD (remove directory) message to a server.

### Usage

```
INT NU_FCP_Client_RMD (FTP_CLIENT *client,  
                      CHAR *path);  
  
INT NU_FCP_Client_XRMD (FTP_CLIENT *client,  
                      CHAR *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the directory to remove.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routines return one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_BAD\_CMD\_SEQUENCE**

The FTP Server returned a “Bad command sequence” error.

- FTP\_INVALID\_USER

The FTP Server did not recognize the user account.

- FTP\_FILE\_UNAVAILABLE

The file on the FTP Server was either not found or the user does not have access to the file.

- FTP\_BAD\_RESPONSE

The FTP Server returned an unknown error code.

- FTP\_INVALID\_PARM

A required parameter is null.

## Description

An error from the server indicates that the path was invalid, that it could not be deleted (under MSDOS, still had files in it), or that removing a directory was a privileged action. Note that the XRMD version is for backward compatibility with older servers that may not recognize the newer forms. The RMD variant calls the XRMD variant if an unrecognized command error is received. The XRMD function need not ever be used and is here for reference only.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path;

/* Connect and log into the FTP Server */
. . .

/* Populate path with the directory to be removed. */
strcpy(path, "deleteme");

/* Send the RMD command */
status = NU_FCP_Client_RMD(&ftp_client, &path);

if (status == NU_SUCCESS)
    /* RMD command was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_RNFR

This function provides a primitive to send an FTP RNFR (rename from) message to the server.

### Usage

```
INT NU_FCP_Client_RNFR (FTP_CLIENT *client,  
                        CHAR          *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the original name.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the file.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Description

An error from the server indicates that the file is not accessible, the file does not exist, or that the command is not allowed at all. A successful reply indicates that a RNT0 command must be sent as the next command message.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path;

/* Connect and log into the FTP Server. */
. . .

/* Populate the path with the filename to be renamed.*/
strcpy(path, "oldname.txt");

/* Send the RNFR command. */
status = NU_FCP_Client_RNFR(&ftp_client, &path);

if (status == NU_SUCCESS)
    /* RNFR was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_RNTO

This function provides a primitive to send an FTP RNTO (rename to) message to the server. This function returns a failure if it is not immediately preceded by a RNFR command.

### Usage

```
INT NU_FCP_Client_RNTO (FTP_CLIENT *client,  
                        CHAR          *path);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the string containing the new name.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_BAD\_CMD\_SEQUENCE**  
The FTP Server returned a “Bad command sequence” error.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_INVALID\_FILE\_NAME**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path;

/* Connect and log into the FTP Server. */
. . .

/* Populate the path with the filename to be renamed.*/
strcpy(path, "oldname.txt");

status = NU_FCP_Client_RNFR(&ftp_client, &path);

if (status == NU_SUCCESS)
{
    /* RNFR successful, so populate the path with the new filename. */
    strcpy(path, "newname.txt");

    /* Send the RNTO command. */
    status = NU_FCP_Client_RNTO(&ftp_client, &path);

    if (status == NU_SUCCESS)
        /* RNTO was successfully sent. */
}
}
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client\_STAT

This function provides a primitive to send an FTP STAT (status) message to the server.

### Usage

```
INT NU_FCP_Client_STAT (FTP_CLIENT *client,  
                        CHAR      *path,  
                        CHAR      *buffer,  
                        INT       bufsize);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **path**  
Pointer to the character string containing the path to the object whose status is desired.
- **buffer**  
Pointer to the storage location for status information.
- **bufsize**  
Size of the buffer.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.

- **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that the requested command has not been implemented.
- **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

The path parameter is a pointer to a character string containing a specification for the file(s) for which a status is requested. If empty, the request is for system information.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        path, buffer;
INT         buff_size;

/* Connect and log into the FTP Server. */
. . .

/* Populate path with the filename for which status is required. */
strcpy(path, ""); /* Request general status */

/* Clear buffer and buff_size. */
strcpy(buffer, "");
buff_size = 0;

/* Send the STAT command. */
status = NU_FCP_Client_STAT(&ftp_client, &path, &buffer, buff_size);

if (status == NU_SUCCESS)
    /* Status successfully sent, and the result is in buffer. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_STOR

This function provides a primitive to send an FTP STOR message to the server.

### Usage

```
INT NU_FCP_Client_STOR (FTP_CLIENT *client,  
                        CHAR *filespec);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **filespec**  
Pointer to the string containing the name of the file to be sent.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_FILE\_UNAVAILABLE**  
The file on the FTP Server was either not found or the user does not have access to the file.
  - **FTP\_MEMORY**  
A memory allocation error occurred.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
- **FTP\_INVALID\_FILE\_NAME**  
The FTP Server could not find the requested file.
- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

## Description

If a path is included in the string, it refers to the path of the destination file rather than the path of the local file. Wildcards are not allowed.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        filename;

/* Connect and log into the FTP Server. */
. . .

/* Populate path with the filename to be stored. */
strcpy(filename, "myfile.txt");

/* Send the STOR command.*/
status = NU_FCP_Client_STOR(&ftp_client, &filename);

if (status == NU_SUCCESS)
    /* Store file command was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_STRU

This function provides a primitive to send an FTP STRU message to the server. The only supported structure type is FILE.

### Usage

```
INT NU_FCP_Client_STRU (FTP_CLIENT *client,  
                        INT          structure);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- structure  
Code for transfer structure type.

### Return Values

- NU\_SUCCESS  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - FTP\_INVALID\_STRU\_CODE  
The structure parameter is invalid.
  - FTP\_STACK\_ERROR  
A send or receive command returned an error while communicating with the FTP Server.
  - FTP\_SERVICE\_UNAVAILABLE  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - FTP\_CMD\_UNRECOGNIZED  
The FTP Server did not recognize the command.
  - FTP\_BAD\_CMD\_FORMAT  
The FTP Server did not recognize the command format.
  - FTP\_CMD\_NOT\_IMPLEMENTED  
The FTP Server replied that requested command has not been implemented.
  - FTP\_INVALID\_USER  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
INT         structure = FTPC_STRU_FILE;

/* Connect and log into the FTP Server. */
. . .

/* Send the Structure command. */
status = NU_FCP_Client_STRU(&ftp_client, structure);

if (status == NU_SUCCESS)
    /* STRU was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_SYST

This function provides a primitive to send an FTP SYST (system information) message to the server.

### Usage

```
INT NU_FCP_Client_SYST (FTP_CLIENT *client,  
                        CHAR        *buffer,  
                        INT         bufsize);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
Pointer to the storage location for the system information.
- **bufsize**  
Size of the buffer.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the ftp server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_FILE\_UNAVAILABLE**

The file on the FTP Server was either not found or the user does not have access to the same.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        buffer;
INT         buff_size;

/* Connect and log into the FTP Server. */
. . .

/* Perform the required actions on FTP Server. */
. . .

/* Clear buffer and buff_size. */
strcpy(buffer, "");
buff_size = 0;

/* Send the System Information command. */
status = NU_FCP_Client_SYST(&ftp_client, &buffer, buff_size);

if (status == NU_SUCCESS)
    /* SYST successfully sent, and the result is in buffer. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Client-Tran\_Ack

This general-purpose function performs the final cleanup following any FTP command that required a data connection. The `ignore_flag` parameter is set if an error condition occurred during the transfer on the client side. This allows the function to be used to clean the final reply message.

### Usage

```
INT NU_FCP_Client-Tran_Ack (FTP_CLIENT *client,  
                           INT          ignore_flag);
```

### Arguments

- `client`  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- `ignore_flag`  
Flag to determine whether or not to use the incoming message.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - `FTP_STACK_ERROR`  
A send or receive command returned an error while communicating with the FTP Server.
  - `FTP_SERVICE_UNAVAILABLE`  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - `FTP_FILE_UNAVAILABLE`  
The file on the FTP Server was either not found or the user does not have access to the file.
  - `FTP_TRANSFER_ABORT`  
The server returned a ‘Transfer Abort’ message.
  - `FTP_SYNTAX_ERROR`  
The server returned a ‘Syntax Error’ message.
  - `FTP_FILE_NOT_FOUND`  
The FTP Server could not find the requested file.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;  
FTP_CLIENT  ftp_client;  
INT         ignore_flag;  
  
/* Connect and log into the FTP Server. */  
. . .  
  
/* Start a file transfer. */  
. . .  
  
/* Send the Transmit Acknowledgement command. */  
status = NU_FCP_Client-Tran_Ack(&ftp_client, ignore_flag);  
  
if (status == NU_SUCCESS)  
    /* Transmit acknowledgement command was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_TYPE

This function provides a primitive to send an FTP TYPE message to the server. Supported types include ASCII, IMAGE (binary), and LOCAL 8 (identical to IMAGE, but necessary for RFC compliance).

### Usage

```
INT NU_FCP_Client_TYPE (FTP_CLIENT *client,  
                        INT         type);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **type**  
Code for transfer type.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
- **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_INVALID\_TYPE\_CODE**  
The type code parameter is invalid.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_CMD\_NOT\_IMPLEMENTED**  
The FTP Server replied that requested command has not been implemented.
  - **FTP\_INVALID\_USER**  
The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**  
The FTP Server returned an unknown error code.
- **FTP\_INVALID\_PARM**  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
INT         type;

/* Connect and log into the FTP Server. */
. . .

/* Populate the transfer type variable. */
type = FTP_TYPE_IMAGE;

/* Send the Type command. */
status = NU_FCP_Client_TYPE(&ftp_client, type);

if (status == NU_SUCCESS)
    /* Type command was successfully sent. */
```

### Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_USER

This function provides a primitive to send an FTP USER message to the server.

### Usage

```
INT NU_FCP_Client_USER (FTP_CLIENT *client,  
                        CHAR *username);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **username**  
Pointer to the string containing the username.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_NEED\_PASSWORD**  
The FTP Server has not received password information needed to process the requested action.
  - **FTP\_NEED\_ACCOUNT**  
The FTP Server has not received account information needed to process the requested action.
  - **FTP\_SERVICE\_UNAVAILABLE**  
The FTP Server returned a “Service Unavailable” message. This may be a generic reply if the FTP Server is preparing for a shut down.
  - **FTP\_CMD\_UNRECOGNIZED**  
The FTP Server did not recognize the command.
  - **FTP\_BAD\_CMD\_FORMAT**  
The FTP Server did not recognize the command format.
  - **FTP\_INVALID\_USER**

The FTP Server did not recognize the user account.

- **FTP\_BAD\_RESPONSE**

The FTP Server returned an unknown error code.

- **FTP\_INVALID\_PARM**

A required parameter is null.

## Description

This function has multiple ‘success’ codes, including the generic success (requiring no further action), and the ‘need password’ success, where the command was received and processed properly but does not have enough information to establish a connection.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        username;

/* Connect to FTP Server */
. . .

/* Populate username. */
strcpy(username, "fred");

/* Send the USER command. */
status = NU_FCP_Client_USER(&ftp_client, &username);

if (status == NU_SUCCESS)
    /* USER command was successfully sent. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

## NU\_FCP\_Client\_Verify\_Caller

This general-purpose function verifies that the currently running task is the ‘owner’ of the FTP session defined by client.

### Usage

```
INT NU_FCP_Client_Verify_Caller (FTP_CLIENT *client);
```

### Arguments

- client  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - FTP\_INVALID\_CLIENT  
The client did not receive a “220 Service Ready” message from FTP Server.
  - FTP\_INVALID\_TASK  
The task ID of the calling task does not match the task ID in the FTP\_CLIENT structure.
  - FTP\_INVALID\_PARM  
A required parameter is null.

### Example

```
STATUS      status;
FTP_CLIENT  ftp_client;

/* Connect and log into the FTP Server .*/
. . .

/* Verify caller. */
status = NU_FCP_Client_Verify_Caller(&ftp_client);

if (status == NU_SUCCESS)
{
    /* The client structure is valid. */
    status = NU_FCP_Client_Type(&ftp_client, FTP_TYPE_ASCII);

    if (status == NU_SUCCESS)
        /* The server's transfer type was successfully set to ASCII. */
}
}
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)



## NU\_FCP\_Reply\_Read

This general-purpose function retrieves a single valid FTP reply message from the command connection and returns the message to the calling function. Each of the FTP command primitives uses this function to receive the reply from the server.

### Usage

```
INT NU_FCP_Reply_Read (FTP_CLIENT    *client,  
                      unsigned char *buffer,  
                      INT            buffsize,  
                      INT            timeout);
```

### Arguments

- **client**  
Pointer to the [FTP\\_CLIENT](#) data structure, defined in the [FTP Client-Level Defines and Data Structures](#) section of this chapter.
- **buffer**  
Pointer to the destination buffer.
- **buffsize**  
The size of the destination buffer.
- **timeout**  
The timeout value for the read.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns one of the following Nucleus XPROT error codes:
  - **FTP\_TIMEOUT**  
The client timed out waiting for a server response.
  - **FTP\_STACK\_ERROR**  
A send or receive command returned an error while communicating with the FTP Server.
  - **FTP\_BAD\_MSG\_FORMAT**  
The reply received was invalid.
  - **FTP\_REPLY\_BUFFER\_OVERRUN**  
The reply data exceeded buffer size.
  - **FTP\_INVALID\_PARM**

A required parameter is null.

## Description

The function returns a single FTP reply message in the memory space pointed to by buffer, returning no more than bufsize characters. The function waits for timeout timer ticks before returning an error if no data becomes available during that time.

## Example

```
STATUS      status;
FTP_CLIENT  ftp_client;
CHAR        buffer[256], username[12];
INT         buff_size, timeout, reply;

/* Connect to the FTP Server. */
. . .

/* Populate username, password and timeout in seconds. */
strcpy(username, "fred");
timeout = (300 * TICKS_PER_SECOND);

/* Clear buffer and buff_size. */
strcpy(buffer, "USER ");
strcat(buffer, username);
buff_size = strlen(buffer);

/* Append \r and \n to buffer. */
buffer[buff_size] = '\r';
buffer[buff_size] = '\n';
buff_size = buff_size + 2;

/* Send buffer contents to the server. */
. . .

/* Read the FTP Server's response. */
reply = NU_FCP_Reply_Read(&ftp_client, buffer, buff_size,
                        timeout);

if (reply >= 0)
{
    /* The reply was received from server, and the reply is in buffer. */
    /* Convert the contents of buffer to integer, and store in 'reply'. */
    switch (reply)
    {
        case 230:
        case 331:
        {
            /* User was authenticated. Send password. */
            .
            .
            break;
        }

        default:
            /* General error condition */
    }
}
```

```
}  
else  
    /* A socket error occurred. */
```

## Related Topics

[FTP Client-Level Command Primitives](#)

[FTP Client-Level Defines and Data Structures](#)

[FTP\\_CLIENT](#)

# Telnet

Nucleus Telnet is a part of the Nucleus Extended Protocol package. The Nucleus Telnet Server provides access to Nucleus PLUS services over the network. The Nucleus Telnet Client can connect to a standard Telnet server and can exchange information or access services, as necessary. The Nucleus Telnet Server and Client can exist over IPv4, IPv6 or dual-stack IPv4/IPv6.

As with other products in Nucleus Extended Protocol, the Nucleus Telnet Server and Client can operate in an IPv6-only environment. When working in an IPv6-only environment, all IPv4 parts of the software can be excluded, resulting in a smaller footprint of target code.

## Telnet Defines

### TN\_VERSION\_COMP

To make NU\_Telnet\_Send service operate as it did in Release 1.1, change the compatibility macro TN\_VERSION\_COMP in *os/include/networking/telopts.h* to TN\_1\_1:

```
#define TN_VERSION_COMP    TN_1_1.
```

### INCLUDE\_TELNET\_SERVER and INCLUDE\_TELNET\_CLIENT

These macros exist in *os/include/networking/telnet\_cfg.h*. If any of these macros are set to zero, the appropriate functions are excluded from the build. For example, if server components of Telnet are not used, INCLUDE\_TELNET\_SERVER can be set to zero to reduce the target image footprint.

```
#define INCLUDE_TELNET_SERVER 1  
#define INCLUDE_TELNET_CLIENT 1
```

## Telnet Negotiation Tables

The Telnet negotiation tables contain the Telnet negotiation options and are used by the [NU\\_Telnet\\_Do\\_Negotiation](#) API which calls [NU\\_Install\\_Negotiate\\_Options](#) to install all the negotiation options into the nego\_buf according to the command and option tables. These

tables, `client_nego_table[]` and `server_nego_table[]`, are defined in the *negotiat.c* file, and look like this:

```
CHAR client_nego_table[]={
/* the command index, the option */
    DONT_WONT, TO_BINARY,
    DO_WONT, TO_ECHO,
    DO_WILL, TO_SGA,
    WILL, TO_NAWS,
    WILL, TO_TERMTYPE
    END_OF_NEGO_TABLE
};

CHAR server_nego_table[]={
/* the command index, the option */
    DONT_WONT, TO_BINARY,
    DONT_WILL, TO_ECHO,
    DO_WILL, TO_SGA,
    DO, TO_TERMTYPE
    DO, TO_NAWS,
    END_OF_NEGO_TABLE
};
```

The first column lists the commands, the second lists the options. Note that the `nego_table` should always end with `END_OF_NEGO_TABLE`. You can find these macros of commands and options in *telopts.h*. More information on Telnet options, option formatting, and the definition of each command can be found in RFCs 854 and 855.

The table `nego_table[]` is designed for basic usage. This table can be modified to match individual needs. However, the formatting should be followed exactly.

## Telnet Data Structures

This section describes custom data structures used for the [Telnet Service Call Primitives](#) and [Telnet Internal Service Calls](#) APIs.

### NU\_TN\_PARAMETERS

The `NU_TN_PARAMETERS` data structure should be used when an application needs to use parameters of the current telnet session. Accessing the internal `TN_SESSION_STRUCT` is not advised; therefore, `NU_TN_PARAMETERS` can be used to store session parameters and make them publicly available. The `NU_TN_PARAMETERS` is defined in *windat.h*.

```
typedef struct {
    int nu_width,
        nu_height,
        nu_termtype,
        nu_nego_naws,
        nu_nego_termtype,
        nu_nego_sga,
        nu_nego_echo,
```

```
    nu_nego_binary;  
} NU_TN_PARAMETERS;
```

The members of the structure are defined in [Table 10-10](#). Some members of this structure have been omitted as they are used internally.

**Table 10-10. NU\_TN\_PARAMETERS**

Member	Description
nu_width	Width of the window.
nu_height	Number of rows in the window.
nu_termtype	Terminal type for this connection.
nu_nego_naws	The state of the NAWS negotiation.
nu_nego_termtype	The state of the TERMTYPE negotiation.
nu_nego_sga	The state of the SGA negotiation.
nu_nego_echo	The state of the ECHO negotiation.
nu_nego_binary	The state of the BINARY negotiation.

## Related Topics

[Telnet Data Structures](#)

[NU\\_Telnet\\_Get\\_Session\\_Parameters](#)

# Telnet Service Call Primitives

The Nucleus Telnet API service call primitives are simple functions that handle the task of initializing Telnet applications, negotiating with the other side, and building the connection.

- [NU\\_Close\\_And\\_Check\\_Retval](#)
- [NU\\_Install\\_Negotiate\\_Options](#)
- [NU\\_Received\\_Exit](#)
- [NU\\_Telnet\\_Check\\_Connection](#)
- [NU\\_Telnet\\_Client\\_Connect](#)
- [NU\\_Telnet\\_Client\\_Connect2](#)
- [NU\\_Telnet\\_Echo\\_Get](#)
- [NU\\_Telnet\\_Echo\\_Set](#)
- [NU\\_Telnet\\_Free\\_Parameters](#)
- [NU\\_Telnet\\_Init\\_Parameters](#)

- [NU\\_Telnet\\_Send](#)
- [NU\\_Telnet\\_Server\\_Accept](#)
- [NU\\_Telnet\\_Socket](#)
- [NU\\_Telnet\\_Specific\\_Negotiate](#)
- [NU\\_Telnet\\_Start\\_Negotiate](#)

## NU\_Close\_And\_Check\_Retval

This function closes the socket being used for the Telnet session.

### Usage

```
INT NU_Close_And_Check_Retval (INT socket);
```

### Arguments

- socket  
The socket index of the Telnet connection.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- Otherwise, the function returns a Nucleus NET error code.

### Example

```
STATUS status;  
INT      socket;  
  
/* Connect and log into the Telnet Server. */  
. . .  
  
/* Perform the required activities on the server. */  
. . .  
  
/* Close the connection to the server. */  
status = NU_Close_And_Check_Retval(socket);  
  
if (status == NU_SUCCESS)  
    /* Connection to the Telnet Server was successfully closed. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Install\_Negotiate\_Options

This function installs the negotiation options.

### Usage

```
INT NU_Install_Negotiate_Options (CHAR *nego_buf,  
                                CHAR *nego_table,  
                                INT  socket);
```

### Arguments

- `nego_buf`  
Pointer to the buffer where the negotiation options are installed.
- `nego_table`  
Pointer to the table of negotiation options to install.
- `socket`  
The socket associated with the connection.

### Return Values

- The number of bytes installed in the `nego_buf`.

### Description

This function is called by [NU\\_Telnet\\_Do\\_Negotiation](#). It does not need to be called directly. `NU_Install_Negotiate_Options` installs all the negotiation options into the `nego_buf` according to the command and option tables, defined in the [Telnet Negotiation Tables](#) section.

### Example

```
INT      socket, size;  
CHAR     nego_buf[30], nego_table[30];  
  
/* Install the Telnet Negotiation Options. */  
size = NU_Telnet_Install_Negotiate_Options(nego_buf, nego_table,  
                                           socket);  
  
/* Send the Negotiation Options to the target host .*/  
. . .  
  
/* Negotiate different parameters with the target host. */  
. . .
```

### Related Topics

[Telnet Service Call Primitives](#)



## NU\_Received\_Exit

This function validates the exit command from the client.

### Usage

```
INT NU_Received_Exit (CHAR *data,  
                     INT  cnt);
```

### Arguments

- data  
Pointer to the input string to be parsed.
- cnt  
The length of the string to be parsed.

### Return Values

- 0  
No exit command was found.
- 1  
Exit command was found.

### Example

```
STATUS status;  
INT     count;  
CHAR    data[256];  
  
/* Start the Telnet Server. */  
. . .  
  
/* Allow the client to connect to the server and perform actions. */  
. . .  
  
/* Check for the Exit command from the client. */  
status = NU_Received_Exit(data, count);  
  
if (status == 1)  
    /* Exit command received from client. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Check\_Connection

This function checks or closes a Telnet connection.

### Usage

```
INT NU_Telnet_Check_Connection (INT socket,  
                                INT close_it);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **close\_it**  
Flag indicating whether to close the connection or check its status (NU\_TRUE - close, NU\_FALSE - check status.)

### Return Values

- **NU\_TRUE**  
When **close\_it** is non-zero (trying to close the connection) and the socket is closed successfully or not connected, or when **close\_it** is zero (checking the status) and the connection on the socket is closed.
- **NU\_FALSE**  
When **close\_it** is zero (checking the status) and the connection on the socket is opened.
- **NU\_INVALID\_SOCKET**  
The socket descriptor passed in through 'socket' was not a valid socket.

### Example

```
STATUS status;  
INT     socket, close_sck = NU_TRUE, count;  
CHAR    data;  
  
/* Start the Telnet Server. */  
. . .  
  
/* Allow clients to connect to theserver and perform actions. */  
. . .  
  
/* Check for the Exit command from the client. */  
status = NU_Received_Exit(&data, count);  
if (status == 1) /* Exit command received from client */  
{  
    if (NU_Telnet_Check_Connection(socket, close_sck)) == NU_TRUE)  
        /* Client connection was closed successfully. */  
}
```

## Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Client\_Connect

This function was replaced by NU\_Telnet\_Client\_Connect2 and is supported only for backward compatibility. This function opens a connection with a Telnet server over IPv4.

### Usage

```
INT NU_Telnet_Client_Connect (CHAR *server_ip,  
                             CHAR *server_name);
```

### Arguments

- **server\_ip**  
Pointer to the network IPv4 IP address of the server host.
- **server\_name**  
Pointer to the machine name of the server host. Nucleus Telnet utilities do not use the machine table.

### Return Values

- A socket descriptor with a value greater than or equal to zero.  
Upon successful completion.
- Otherwise, the function returns an operating system-specific error code or Nucleus NET error code.

### Example

```
INT      socket;  
CHAR     server_name[20];  
CHAR     server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};  
  
/* Install Negotiation Options on the client. */  
. . .  
  
/* Populate server_name with the appropriate values. */  
strcpy(server_name, "telnet_server");  
  
/* Perform Client Connect. */  
socket = NU_Telnet_Client_Connect(server_ip, server_name);  
  
if (socket >= 0)  
    /* Successfully connected to the server. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Client\_Connect2

This function opens a connection with a Telnet server. (family = NU\_FAMILY\_IP6) makes a dual-stack IPv4/IPv6 connection if INCLUDE\_IPV4 is set to NU\_TRUE in the *os/include/networking/net\_cfg.h* file. Otherwise, it will be an IPv6-only connection.

### Usage

```
INT NU_Telnet_Client_Connect2 (CHAR  *server_ip,
                              CHAR  *server_name,
                              INT16 family);
```

### Arguments

- **server\_ip**  
Pointer to the network IP address of the server host.
- **server\_name**  
Pointer to the machine name of the server host. Nucleus Telnet utilities do not use the machine table.
- **family**  
The network protocol to use for the connection; either NU\_FAMILY\_IP for IPv4 or NU\_FAMILY\_IP6 for IPv6.

### Return Values

- A socket descriptor with a value greater than or equal to zero.  
Upon successful completion.
- Otherwise, the function returns an operating system-specific error code or Nucleus NET error code.

### Example

```
INT  socket;
INT16 server_family
CHAR  server_name[20];
CHAR  server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};

/* Install Negotiation Options on the client. */
. . .

/* Populate the server name. */
strcpy(server_name, "telnet_server");

/* Populate the server family. */
server_family = NU_FAMILY_IP;

/* Perform Client Connect. */
socket = NU_Telnet_Client_Connect2(server_ip, server_name,
                                   server_family);

if (socket >= 0)
```

```
/* Successfully connected to the server */
```

## Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Echo\_Get

This function returns the echo parameters that have been negotiated on a given socket.

### Usage

```
INT NU_Telnet_Echo_Get (INT socket);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.

### Return Values

- The previous echo value.  
Upon successful completion.  
Otherwise, a Nucleus NET error code or the following Nucleus XPROT error code:
  - -1  
The socket is invalid.

### Description

When used in conjunction with `NU_Telnet_Echo_Set()`, echoing can be suppressed for a period of time over a telnet connection. This is useful if the Nucleus Telnet Server should not echo back the password that is being typed by the user, and should suppress the echo for that time interval.

### Example

```
STATUS status;
INT client_socket, old_nego_echo;
/* Start Telnet Server. */
. . .
/* Wait for the client to request a Telnet session, Negotiate params. */
. . .
/* Prompt and get username, prompt for password. */
. . .
/* Save the currently negotiated echo parameters. */
old_nego_echo = NU_Telnet_Echo_Get(client_socket);

if (old_nego_echo > 0)
    /* Existing echo parameters were successfully retrieved. */
    /* Suppress echo for the password input. */
    . . .
    /* Get the password from the client. */
    . . .
    /* Restore the old nego echo settings. */
    . . .
```

## Related Topics

[Telnet Service Call Primitives](#)



## NU\_Telnet\_Echo\_Set

This function is used to inhibit or resume echo over a connection. This is useful because Nucleus Telnet Server should not echo back the password that is being typed and should suppress echo for that time interval.

### Usage

```
STATUS NU_Telnet_Echo_Set (INT socket,  
                          INT new_nego_echo);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **new\_nego\_echo**  
Specifies the new echo parameters to be set.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
  
Otherwise, the routine returns a Nucleus NET error code or the following Nucleus XPROT error code:
  - **-1**  
The socket is invalid

### Description

Before calling this routine, make sure you have saved off the existing negotiated echo parameters with `NU_Telnet_Echo_Get()`. Call this routine with `new_nego_echo` set to `NEGO_I_WONT|NEGO_HE_DONT` to suppress echo over the connection. To restore echo settings, call this routine with the saved negotiated the echo parameters.

### Example

```
STATUS status;  
INT client_socket, old_nego_echo;  
/* Start Telnet Server. */  
. . .  
  
/* Wait for the client to request a Telnet session, Negotiate params. */  
. . .  
  
/* Prompt and get username, prompt for password. */  
. . .  
  
/* Get the existing negotiated echo parameters */  
old_nego_echo = NU_Telnet_Echo_Get(client_socket);  
  
if (old_nego_echo > 0)
```

```
{
    /* Success retrieving echo params. Turn echo off. */
    status = NU_Telnet_Echo_Set(client_socket,
                                NEGO_I_WONT|NEGO_HE_DONT);
    if (status == NU_SUCCESS)
    {
        /* Get the password from the client. */
        . . .

        /* Restore old nego echo settings. */
        status = NU_Telnet_Echo_Set(client_socket, old_nego_echo);

        if (status == NU_SUCCESS)
            /* Old Nego Echo values were successfully restored. */
    }
}
```

## Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Free\_Parameters

This function frees the memory allocated by NU\_Telnet\_Server\_Init\_Parameters or NU\_Telnet\_Client\_Init\_Parameters.

### Usage

```
VOID NU_Telnet_Free_Parameters (INT socket);
```

### Arguments

- **socket**  
The socket associated with the connection.

### Example

```
INT      socket;

/* Start the Telnet Server. */
. . .

/* Allow clients to connect. */
. . .

/* If there was an error negotiating with the client, free the
 * parameters associated with the client connection.
 */
NU_Telnet_Free_Parameters(socket);
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Init\_Parameters

This function initializes the parameters of a Telnet server session.

### Usage

```
STATUS NU_Telnet_Init_Parameters (INT server_socket);
```

In the current release, this function maps to:

```
STATUS NU_Telnet_Server_Init_Parameters (INT server_socket);
```

### Arguments

- `server_socket`

The socket associated with the Telnet server connection.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code.

### Description

The memory for a Telnet server session structure is allocated when its fields are initialized. The array of pointers to Telnet sessions is declared in *windat.h*.

### Example

```
INT      socket;
STATUS   status;

/* Start the Telnet Server. */
. . .

/* Wait for the Telnet client to connect. */
. . .

/* If a client connected to the server, run init params. */
status = NU_Telnet_Init_Parameters(socket);

if (status == NU_SUCCESS)
    /* Init params was successful. Start negotiations. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Send

In Releases 1.1 and earlier, this function sends data to the network during Telnet communication. NU\_Telnet\_Send is different from NU\_Send in that the number of arguments allowed varies. If this feature is not needed, NU\_Send should be used instead.

In Releases 1.2 and later, this function maps to NU\_Send.

### Usage

```
INT NU_Telnet_Send (INT  socket,
                   CHAR *fmt);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **fmt**  
Pointer to the buffer containing the data to be sent.

### Return Values

- The number of bytes sent  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code or Nucleus NET error code.

### Example

```
INT  socket, retval;
CHAR buffer[256];

/* Install Negotiation Options on the client. */
. . .

/* Wait for clients to connect. */
. . .

/* Build welcome message in buffer to send to the client. */
strcpy(buffer, "Welcome to Nucleus Telnet Server\r\n");

/* Send a message to the client. */
retval = NU_Telnet_Send(socket, buffer);

if (retval >= 0)
    /* The message successfully sent to the client. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Server\_Accept

This function accepts a connection from a Telnet client.

### Usage

```
INT NU_Telnet_Server_Accept (INT socket);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.

### Return Values

- A socket index  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code or Nucleus NET error code.

### Example

```
INT      socket, retval;

/* Start the Telnet Server. */
. . .

/* Wait for clients to connect. */
. . .

/* The client requests a Telnet session. Accept it. */
retval = NU_Telnet_Server_Accept(socket);

if (retval >= 0)
    /* Client session was accepted successfully. */
```

### Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Socket

This function creates the first socket for the telnet connection.

### Usage

```
INT NU_Telnet_Socket (CHAR *server_ip,  
                     CHAR *server_name);
```

### Arguments

- **server\_ip**  
Pointer to the Telnet server IP address. If the server IP address is 0.0.0.0 the routine will use any IP address.
- **server\_name**  
Pointer to the Telnet server name.

### Return Values

- The socket index  
Upon successful completion.
- Otherwise, the routine returns the following Nucleus XPROT error code:
  - -1  
A general error has occurred. This might be due to an operating system error or a Nucleus NET error.

### Description

If IPv6 is included in Nucleus NET and the `server_ip` is NULL, the socket type is IPv6; otherwise, the socket type is IPv4. If the created socket type is IPv6, it processes both IPv4 and IPv6 connections.

### Example

```
INT    socketd;  
CHAR   server_name[20];  
CHAR   server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};  
  
/* Get the Telnet Server name. */  
strcpy(server_name, "tn-server");  
  
/* Get the first template socket for the server. */  
socketd = NU_Telnet_Socket(server_ip, server_name);  
  
if (socketd >= 0)  
    /* Template socket was created successfully. */
```

## Related Topics

[Telnet Service Call Primitives](#)



## NU\_Telnet\_Specific\_Negotiate

This function negotiates window size (terminal screen) and sets bits of nego\_NAWS of the global TN\_Session structure. This function is outdated and is left for compatibility purposes only. It is not recommended to use it.

### Usage

```
INT NU_Telnet_Specific_Negotiate (INT socket,  
                                  INT option);
```

### Arguments

- **socket**  
The socket index of this connection.
- **option**  
The telnet option for this connection.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.  
Otherwise, the routine returns an operating system-specific error code or Nucleus NET error code.
- **NO\_NEGO\_OPTION\_INSTALLED**  
If no windows size negotiation option was installed in the global client\_nego\_table or server\_nego\_table.

### Description

The following sequences of commands supersede this function and should be used instead:

For Telnet client:

```
STATUS status;  
  
status = NU_Telnet_Client_Init_Parameters(client_socket);  
  
if (status != NU_SUCCESS)  
    /* error initializing client_socket */  
else  
    status =  
        NU_Telnet_Do_Negotiation(client_socket, client_nego_table);
```

For Telnet server:

```
STATUS status;  
  
status = NU_Telnet_Server_Init_Parameters(server_socket);  
  
if (status != NU_SUCCESS)
```

```
        /* error initializing server_socket */  
    else  
        status =  
            NU_Telnet_Do_Negotiation(server_socket, server_nego_table);
```

nego\_NAWS is defined in *windat.h*, and NEGOT\_I\_DO, NEGOT\_I\_DONT, NEGOT\_I\_WILL, NEGOT\_I\_WONT, NEGOT\_HE\_DO, NEGOT\_HE\_DONT, NEGOT\_HE\_WILL, NEGOT\_HE\_WONT are constants defined in *telopts.h*. For further information, refer to RFC 854.

## Example

```
STATUS status;  
INT      socket, option;  
  
/* Start the Telnet Server. */  
. . .  
  
/* Wait for the client to request a Telnet session. */  
. . .  
  
/* Install Negotiation options. */  
. . .  
  
/* Start Window Size negotiation. */  
status = NU_Telnet_Specific_Negotiate(socket, option);  
  
if (status == NU_SUCCESS)  
    /* Window size was negotiated successfully. */
```

## Related Topics

[Telnet Service Call Primitives](#)

## NU\_Telnet\_Start\_Negotiate

This function starts a Telnet negotiation.

### Caution



This function is outdated and is left for compatibility purposes only. It is not recommended to use it. The following sequences of commands supersede this function and should be used instead:

For Telnet client:

```
STATUS status;

status = NU_Telnet_Client_Init_Parameters(client_socket);

if (status != NU_SUCCESS)
    /* error initializing client_socket */
else
    status =
        NU_Telnet_Do_Negotiation(client_socket, client_nego_table);
```

For Telnet server:

```
STATUS status;

status = NU_Telnet_Server_Init_Parameters(server_socket);

if (status != NU_SUCCESS)
    /* error initializing server_socket */
else
    status =
        NU_Telnet_Do_Negotiation(server_socket, server_nego_table);
```

## Usage

```
VOID NU_Telnet_Start_Negotiate (INT  socket,
                                CHAR  *nego_table);
```

## Arguments

- **socket**  
Specifies the socket index of this connection.
- **nego\_table**  
A pointer to an array of elements, the first element of each pair is the index of the Telnet negotiation command, the second is the Telnet negotiation option. See [“NU\\_Install\\_Negotiate\\_Options”](#) on page 2114 for an example of nego\_table.

## Example

```
INT      socketd;
CHAR     nego_table[30];
```

```
/* Start the Telnet Server. */  
. . .  
  
/* Wait for the client to request a Telnet session. */  
. . .  
  
/* Populate the negotiation table with initial values. */  
strcpy(nego_table, *server_nego_table);  
  
/* Start negotiations with the client. */  
NU_Telnet_Start_Negotiate(socket, nego_table);
```

## Related Topics

[Telnet Service Call Primitives](#)

## Telnet Internal Service Calls

The Nucleus Telnet API internal service calls are functions that handle the details of the communication according to the Telnet protocol (see RFC 854). The functions discussed in this section include a brief description, the functions' syntax, a description of any parameters used, and return values.

- [NU\\_Receive\\_NVT\\_Key](#)
- [NU\\_Send\\_NVT\\_Key](#)
- [NU\\_Telnet\\_Client\\_Init\\_Parameters](#)
- [NU\\_Telnet\\_Do\\_Negotiation](#)
- [NU\\_Telnet\\_Get\\_Filtered\\_Char](#)
- [NU\\_Telnet\\_Get\\_Session\\_Parameters](#)
- [NU\\_Telnet\\_Parse](#)
- [NU\\_Telnet\\_Pick\\_Up\\_Ascii](#)
- [NU\\_Telnet\\_Server\\_Init\\_Parameters](#)
- [NU\\_Wait\\_For\\_Pattern](#)

## NU\_Receive\_NVT\_Key

This function converts a received key mapping code to the corresponding local key value (according to the pre-defined mapping table declared in *vtkeys.h*), and explains it, or executes the function that it represents.

### Usage

```
INT NU_Receive_NVT_Key (INT  socket,
                       CHAR  *string,
                       CHAR  execute);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **string**  
Pointer to the key code string converted.
- **execute**  
Flag that determines if the function that the key value represents is executed.

### Return Values

- The number of bytes the string contains:  
Upon successful completion.
- 0  
Otherwise.

### Description

The Nucleus Telnet Utilities cannot be designed to cover all terminal types. Local key values are dependent on your device and application program. This table can be modified, expanded, or more tables can be added to suit the environment. You may need to add function calls to do what each specific key invokes.

### Example

```
INT      socket, retval;
CHAR     string[256], execute;

/* Start Telnet server, and wait for clients to connect. */
. . .

/* Parse the incoming string from the client. */
. . .

/* Check to see if the NVT Key was received. */
retval = NU_Received_NVT_Key(socket, string, execute);

if (retval > 0)
    /* The received string maps to a vt key. */
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Send\_NVT\_Key

This function sends a local keystroke to the remote host. If the local key value is listed in the mapping table pre-defined in *vtkeys.h*, it is converted to the terminal emulation key string according to the table. Then the string is sent to the remote host.

### Usage

```
VOID NU_Send_NVT_key (INT      socket,  
                     UINT32   key_val);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **key\_val**  
The value of local key.

### Description

Nucleus Telnet utilities cannot be designed to cover all terminal types. Local key values are dependent on your device and how your application program produces it. This table can be modified, expanded, or more tables can be added according to the environment.

### Example

```
INT      socket;  
UINT32   key;  
  
/* Connect to a Telnet Server. */  
. . .  
  
/* Send a character, ASCII value 65 = 'A'. */  
key = 65;  
  
/* Send a NVT Key to the server. */  
NU_Send_NVT_Key(socket, key);
```

### Related Topics

[Telnet Internal Service Calls](#)



## NU\_Telnet\_Client\_Init\_Parameters

This function initializes parameters of a client Telnet session. The Telnet client should always call it after calling the `NU_Telnet_Client_Connect()` (which creates a client socket and connects to the telnet server) and before calling `NU_Telnet_Do_Negotiation()`.

### Usage

```
STATUS NU_Telnet_Client_Init_Parameters (INT client_socket);
```

### Arguments

- `client_socket`  
Specifies the client socket index of this connection.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code.

### Example

```
INT      socket;
STATUS   status;
CHAR     server_ip[] = {(CHAR) 10, (CHAR) 0, (CHAR) 0, (CHAR) 1};
Extern CHAR client_nego_table[];

/* Open the client socket and connect to the server. */
socket = NU_Telnet_Client_Connect(server_ip, "tn-server");

if (socket < 0)
    return;

/* Before negotiation, always call the relevant _Init_Parameters()
 * function. It sets a flag showing if we are server / client.
 */
if (NU_Telnet_Client_Init_Parameters(client_socket) != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(client_socket);
    NU_Close_Socket(client_socket);
    return;
}

/* telnet client options negotiation */
if (NU_Telnet_Do_Negotiation(client_socket, client_nego_table)
    != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(client_socket);
    NU_Close_Socket(client_socket);
    return;
}
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Telnet\_Do\_Negotiation

This function represents a Telnet options negotiation driver. It installs negotiation options calling [NU\\_Install\\_Negotiate\\_Options](#), sends them to the other side, and calls other functions responsible for negotiation of Telnet options.

---

**Note**

In the current release, the following negotiation RFCs are supported: RFC856 (Telnet Binary Transmission), RFC857 (Telnet Echo Option), RFC858 (Telnet Suppress Go Ahead Option), RFC1073 (Telnet Window Size Option), RFC1091 (Telnet Terminal-Type Option).

---

### Usage

```
STATUS NU_Telnet_Do_Negotiation (INT  socket,
                                CHAR  *nego_table);
```

### Arguments

- `socket`  
Specifies the socket index of this connection.
- `nego_table`  
Pointer to the `client_nego_table` or `server_nego_table`, defined in the [Telnet Negotiation Tables](#) section.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error.

### Example

For Telnet client:

```
STATUS status;

status = NU_Telnet_Client_Init_Parameters(client_socket);

if (status == NU_SUCCESS)
    status = NU_Telnet_Do_Negotiation(client_socket,
                                     client_nego_table);

else /* error initializing client_socket, quit */
```

For Telnet server:

```
STATUS status;

status = NU_Telnet_Server_Init_Parameters(server_socket);
```

```
if (status == NU_SUCCESS)
    status = NU_Telnet_Do_Negotiation(server_socket,
                                     server_nego_table);

else /* error initializing server_socket, quit */
```

## Related Topics

[Telnet Internal Service Calls](#)

[Telnet Negotiation Tables](#)

[NU\\_Install\\_Negotiate\\_Options](#)

## NU\_Telnet\_Get\_Filtered\_Char

This function gets one character from the network. If the incoming byte is the first byte of the string of a Telnet command or option, or is the string of a terminal emulation keystroke code, this function continues to collect characters. The resulting string is then passed to the appropriate function for processing. If the parameter timeout is set, this service waits until it gets one character, and returns it to the caller. If the parameter timeout is not set, it simply checks the network buffer without waiting.

### Usage

```
STATUS NU_Telnet_Get_Filtered_Char (INT      socket,  
                                  UNSIGNED timeout,  
                                  CHAR      *retchar);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **timeout**  
Timeout value in ticks. Specifies how long the function needs to wait for incoming data. Two other possible values are NU\_SUSPEND (wait indefinitely) and NU\_NO\_SUSPEND (return immediately if data is not available).
- **retchar**  
This is where the function stores the character read in case of success.

### Return Values

- **NU\_SUCCESS** and an ASCII character in the *\*retchar* parameter  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code or Nucleus NET error code.

### Example

```
STATUS  status;  
INT     socket;  
UNSIGNED timeout;  
CHAR    chr;  
  
/* Connect to the Telnet Server. */  
. . .  
  
/* Set character to scan for and set timeout values. */  
chr = ':';  
timeout = (90 * TICKS_PER_SECOND)  
  
/* Get a filtered character from the server. */  
status = NU_Telnet_Get_Filtered_Char(socket, timeout, &chr);  
  
if (status == NU_SUCCESS)
```

```
/* Character received from the server */
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Telnet\_Get\_Session\_Parameters

This function gets the Telnet session parameters from the internal TN\_Session structure.

### Usage

```
STATUS NU_Telnet_Get_Session_Parameters (
                                INT          socket,
                                NU_TN_PARAMETERS *session_params);
```

### Arguments

- **socket**  
Specifies the socket index of this connection.
- **session\_params**  
Pointer to the structure where Telnet session parameters are stored.  
Data structure [NU\\_TN\\_PARAMETERS](#) is defined in the [Telnet Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion. Otherwise, the routine returns an operating system-specific error code.

### Description

Because it is not recommended for an application to access the internal TN\_Session structure directly, NU\_Telnet\_Get\_Session\_Parameters() should be used to copy the Telnet session parameters from the internal TN\_SESSION to the publicly available [NU\\_TN\\_PARAMETERS](#).

### Example

```
NU_TN_PARAMETERS session_params;
INT server_socket, client_window_width, client_window_height;
extern CHAR server_nego_table[];

/* Initialize the server socket. */
NU_Telnet_Server_Init_Parameters(server_socket);

/* Negotiate with the client. */
NU_Telnet_Do_Negotiation(server_socket, server_nego_table);

/* Get the results of negotiation. */
NU_Telnet_Get_Session_Parameters(server_socket, &session_params);

/* Check if client sent his windows size. */
if ((session_params.nego_NAWS & NEGO_I_DO) &&
    (session_params.nego_NAWS & NEGO_HE_WILL))
{
    client_window_width = session_params.nu_width;
    client_window_height = session_params.nu_height;
}
```

## Related Topics

[Telnet Internal Service Calls](#)

[Telnet Data Structures](#)

[NU\\_TN\\_PARAMETERS](#)



## NU\_Telnet\_Parse

This function parses the incoming data from the other end of a Telnet connection. It recognizes the Telnet commands and options and executes them automatically.

### Usage

```
INT NU_Telnet_Parse (INT      socket,  
                    UINT8    *st,  
                    INT      cnt,  
                    INT      terminal);
```

### Arguments

- **socket**  
The socket index of this connection.
- **st**  
Pointer to the buffer where the received data is stored.
- **cnt**  
The length of the data buffer.
- **terminal**  
Specifies whether data other than Telnet commands and Telnet options are automatically dumped to the users terminal screen when one is present.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.

### Example

```
STATUS status;  
INT      socket, count, terminal = 1;  
UINT8    buffer[256];  
  
/* Start Telnet Server. */  
. . .  
  
/* Wait for clients to connect. */  
. . .  
  
/* Start negotiations. */  
. . .  
  
/* Parse result of negotiations. */  
status = NU_Telnet_Parse(socket, buffer, count, terminal);  
  
/* Return status is always NU_SUCCESS. */
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Telnet\_Pick\_Up\_Ascii

This function picks up only ASCII code bytes from the incoming data. It filters out all Telnet commands and options, terminal emulation key codes, and ESC characters. It collects the rest of the data in the original buffer.

### Usage

```
INT NU_Telnet_Pick_Up_Ascii (UINT8 *st,  
                             INT    cnt);
```

### Arguments

- `st`  
Pointer to the data buffer.
- `cnt`  
Specifies how many bytes of data are in the buffer pointed to by `st`.

### Return Values

- `NU_SUCCESS`  
Upon successful completion.
- Otherwise, the routine returns one of the following codes:
  - 1  
Sub-negotiation codes
  - 2  
Telnet command and option
  - 4  
ASCII codes
  - 8  
ESC string

### Example

```
STATUS status;  
INT    count;  
UINT8  buffer[256];  
/* Start Telnet Server. */  
. . .  
  
/* Wait for client to connect, then perform negotiations. */  
. . .  
  
/* Pick up ASCII from client. */  
status = NU_Telnet_Pick_Up_Ascii(buffer, count);
```

```
if (status == NU_SUCCESS)
    /* Pick up ASCII was successfully completed. */
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Telnet\_Server\_Init\_Parameters

This function initializes parameters of a server Telnet session. The Telnet server should always call this function after calling the NU\_Telnet\_Socket() and NU\_Telnet\_Server\_Accept() and before calling NU\_Telnet\_Do\_Negotiation().

### Usage

```
STATUS NU_Telnet_Server_Init_Parameters (INT server_socket);
```

### Arguments

- server\_socket  
Specifies the server socket index of this connection.

### Return Values

- NU\_SUCCESS  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code.

### Example

```
INT socket, server_socket;
CHAR server_name[12];

do    /* Create a telnet server socket template. */
{
    socket = (INT16) (NU_Telnet_Socket(SERVER_Ip, server_name));
}while ( socket < 0 );

/* Wait for a telnet client. */
server_socket = (INT16) (NU_Telnet_Server_Accept(socket));

if (socket < 0)
    /* error */;

/* Initialize the data structure that contains all the
 * parameters of the telnet server session. */
if (NU_Telnet_Server_Init_Parameters(server_socket) != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(server_socket);
    NU_Close_Socket(server_socket);
    return;
}

/* telnet server options negotiation */
if (NU_Telnet_Do_Negotiation(server_socket,
                             server_nego_table) != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(server_socket);
    NU_Close_Socket(server_socket);
    return;
}
```

## Related Topics

[Telnet Internal Service Calls](#)

## NU\_Wait\_For\_Pattern

This function reads from the socketd one character at a time and places the characters to \*buf until it finds the pattern, or runs out of space in \*buf.

### Usage

```
STATUS NU_Wait_For_Pattern (INT          socketd,  
                           CHAR          *buf,  
                           UINT16        buf_size,  
                           UINT16        *buf_chars_read,  
                           CONST CHAR    *pattern,  
                           UINT16        pattern_len,  
                           UINT16        *pattern_chars_read,  
                           UNSIGNED      timeout);
```

### Arguments

- **socketd**  
Specifies the socket index of this connection.
- **buf**  
Pointer to buffer to store chars read from socketd.
- **buf\_size**  
Size of the buf argument.
- **buf\_chars\_read**  
Number of characters read from the socketd and placed to buf.
- **pattern**  
Pointer to the string for which we are waiting/looking. May be any sequence of chars including Nulls. For example, the search pattern `pattern[] = {0,0,'a','b','c',0,0}`; (“abc” preceded and followed by two Nulls) is a perfectly legal search pattern.
- **pattern\_len**  
Length of the pattern. If you try to match one or more Nulls, `strlen(pattern)` does not work. For example, in order to correctly match `pattern[] = {0,0,'a','b','c',0,0}`; `pattern_len` should be set to 7.
- **pattern\_chars\_read**  
Pointer to how many chars in the end of buf match beginning of pattern. Used internally by `NU_Wait_For_Pattern()`, but should always be set to zero by the caller before starting to match for a new pattern.
- **timeout**  
Timeout value in ticks. Two other possible values are `NU_SUSPEND` (wait indefinitely) and `NU_NO_SUSPEND` (return immediately if data is not available).

## Return Values

- **NU\_TRUE** and `buf_chars_read` indicates the number of characters read into the buffer.  
Upon successful completion.
- Otherwise, the routine returns one of the following error codes:
  - **NU\_FALSE**  
Pattern not found. `buf_chars_read` indicates how many characters were read into the buffer.
  - **-1**  
One of the input parameters is invalid.
  - **NU\_NO\_DATA**  
Timed out. Returned only when timeout argument is not **NU\_SUSPEND** or **NU\_NO\_SUSPEND**. When **NU\_NO\_DATA** is returned, check `buf_chars_read` to see if any data was read into the buffer.

## Description

Once the pattern is found, the function returns **NU\_TRUE**. The caller can check the `buf_chars_read` to see how many bytes/chars were read before the pattern was found. The first character of the pattern in this case is placed in `buf[buf_chars_read]`. If the pattern is not found, the function returns **NU\_FALSE**. The `buf_chars_read` indicates how many characters were read into the buf.

### Note



`pattern_chars_read` is used internally by `NU_Wait_For_Pattern()`. It shows how many chars in the end of `*buf` match the pattern. It is essential that the caller set `pattern_chars_read` to 0 before calling `NU_Wait_For_Pattern()` for the very first time, and NOT change it unless the pattern was found, or search for a new pattern has started.

Every time before calling `NU_Wait_For_Pattern()` to start looking for a new pattern, `pattern_chars_read` should be set to zero. After the pattern is found, `pattern_chars_read == pattern_len`.

## Example

```
CHAR    buf[256], server_name[20], server_ip[4];
UINT16  buf_chars_read = 0, pattern_chars_read = 0;

const   char login_pattern[]="login: ";
const   UINT16 login_pattern_len = strlen(login_pattern);

const   char password_pattern[]="Password: ";
const   UINT16 password_pattern_len = strlen(password_pattern);

const   UINT16 sleep_time = 200;
INT8    cmd1[] = {'l', 'a', 's', 't', ' ', '-', '2', '0', CR, LF, 0};
```



```
/* Populate Server IP and name. */
server_ip[] = {(char)10, (char)0, (char)0, (char)1};
strcpy(server_name, "tn-server");

socket = NU_Telnet_Client_Connect(&server_ip, &server_name);
if (socket < 0)
    return;
if (NU_Telnet_Client_Init_Parameters(socket) != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(socket);
    NU_Close_Socket(socket);
    return;
}

/* telnet client options negotiation */
if (NU_Telnet_Do_Negotiation(socket, client_nego_table)
    != NU_SUCCESS)
{
    NU_Telnet_Free_Parameters(socket);
    NU_Close_Socket(socket);
    return;
}

/* Read until we get the server's "Login:" prompt . . . */
while((ret_value = NU_Wait_For_Pattern(socket, buf, 256,
                                     &buf_chars_read,
                                     login_pattern,
                                     login_pattern_len,
                                     &pattern_chars_read,
                                     300)) != NU_TRUE)
{
    if ( (ret_value == NU_NO_DATA) ||
        (ret_value == NU_NOT_CONNECTED) ); /* error */
}

/* Send the login name. */
NU_Send(socket, login_name, strlen(login_name), 0);
NU_Sleep(5 * TICKS_PER_SECOND);

buf_chars_read = pattern_chars_read = 0;

/* Read input until we get the server's "Password:" prompt . . . */
while((ret_value = NU_Wait_For_Pattern(socket, buf, 256,
                                     &buf_chars_read,
                                     password_pattern,
                                     password_pattern_len,
                                     &pattern_chars_read,
                                     NU_NO_SUSPEND)) != NU_TRUE)
{
    if ( (ret_value == NU_NO_DATA) ||
        (ret_value == NU_NOT_CONNECTED) ); /* error */
}

/* Send the password. */
NU_Send(socket, password, strlen(password), 0);
NU_Sleep(5 * TICKS_PER_SECOND);
```

```
/* Now, send some UNIX commands to the server . . . */  
NU_Send(socket, command1, strlen(command1), 0);
```

## Related Topics

[Telnet Internal Service Calls](#)

# TFTP Server

The Nucleus TFTP Server supports OCTET (binary) mode, as discussed within RFC 1350. It processes read and write requests from a TFTP Client over IPv4 and IPv6. The TFTP Server, depending on the request, reads data from or stores data into the file system.

As with other products in Nucleus XPROT, the Nucleus TFTP Server can operate in an IPv6-only environment. When working in an IPv6-only environment, all IPv4 parts of the software can be excluded, resulting in a smaller footprint of target code.

The Nucleus TFTP Server can be stopped and restarted without disrupting other services running in the system. When the server is stopped, it waits for the current client to complete the file transfer operation, and then shuts down the TFTP service.

Nucleus Networking 5.2 and later versions provide a mechanism for services to send messages and advanced debugging information to applications. This mechanism is called the Notification Module. The TFTP Server uses the Notification Module to send debugging information during termination sequences. The application that requested the TFTP Server to stop can query the debug queue to determine if the server was stopped.

## TFTP Server Defines and Data Structures

### TFTP Server Defines

The Nucleus TFTP Server uses several constants in the form of `#defines`. [Table 10-11](#) is a list of these defines, their values, and in what category they belong (whether they are a return code, internally used values, or parameter values).

**Table 10-11. Nucleus TFTP Server Defines**

Name	Value	Category
TFTP_RRQ_OPCODE	1	Parameter value
TFTP_WRQ_OPCODE	2	Parameter value
TFTP_DATA_OPCODE	3	Parameter value
TFTP_ACK_OPCODE	4	Parameter value

**Table 10-11. Nucleus TFTP Server Defines (cont.)**

Name	Value	Category
TFTP_ERROR_OPCODE	5	Parameter value
TFTP_OACK_OPCODE	6	Parameter value
TFTP_ERROR	-1751	Return code
TFTP_FILE_NFOUND	-1752	Return code
TFTP_ACCESS_VIOLATION	-1753	Return code
TFTP_DISK_FULL	-1754	Return code
TFTP_FILE_ERROR	-1755	Return code
TFTP_BAD_OPERATION	-1756	Return code
TFTP_UNKNOWN_TID	-1757	Return code
TFTP_FILE_EXISTS	-1758	Return code
TFTP_NO_SUCH_USER	-1759	Return code
TFTP_BAD_OPTION	-1760	Return code
TFTP_NO_MEMORY	-1761	Return code
TFTP_CON_FAILURE	-1762	Return code
TFTPS_DUPLICATE_DATA	-1763	Return code
TFTPS_DUPLICATE_ACK	-1764	Return code
TFTPS_SERVER_TERMINATED	-1765	Return code
TRANSFERRING_FILE	100	Internally used value
TRANSFER_COMPLETE	102	Internally used value
TFTP_BLOCK_SIZE_DEFAULT	512	Internally used value
TFTP_HEADER_SIZE	4	Internally used value
TFTP_ACK_SIZE	4	Internally used value
TFTP_BLOCK_SIZE_MAX	65464	Internally used value
TFTP_BLOCK_SIZE_MIN	8	Internally used value
TFTP_BUFFER_SIZE_MIN	50	Internally used value
TFTP_PARSING_LENGTH	128	Internally used value
READ_TYPE	0	Internally used value
WRITE_TYPE	1	Internally used value
TFTP_NUM_RETRANS	3	Internally used configurable value
TFTP_TIMEOUT_DEFAULT	30	Internally used configurable value

## TFTP Server Data Structures

There are no data structures specific to the TFTP API functions.

## TFTP Accepting and Processing Options

RFC 2347 defines the client and server behavior in using options, and RFC 2348 and 2349 define a set of options that a client can send with a GET or PUT request. These options include:

**Table 10-12. TFTP Options**

Option	Description
Transmission Size	The total size of the file to be transmitted.
Block Size	The size of each individual data block to transmit.
Timeout	The server's timeout value.

The Nucleus TFTP Server supports all of the options in [Table 10-12](#). You may, however, be able to increase performance and reduce memory usage by not processing incoming options.

## Turning Off Options Processing

The macro `TFTPS_RFC2347_COMPLIANT` in the `os/include/networking/tftps_cfg.h` file allows you to remove options processing from the TFTP Server code. Set this macro to `NU_FALSE` to disable options processing and `NU_TRUE` to enable options processing.

By default, this macro is set to `NU_FALSE`.

## TFTP Server Functions

- [TFTP\\_Server\\_Init](#)
- [TFTP\\_Server\\_Uninit](#)

## TFTP\_Server\_Init

This function initializes the TFTP\_Server\_Task.

### Usage

```
STATUS TFTP_Server_Init (CHAR *dv_name);
```

### Arguments

- **dv\_name**  
Pointer to the name of the interface that processes the incoming TFTP requests. A value of NULL causes the TFTP Server to process incoming requests on all interfaces.

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- Otherwise, the routine returns an operating system-specific error code or Nucleus NET error code.

### Description

This is the external interface to start the TFTP service within your application. The parameter **dv\_name** is passed in after the initialization of your network devices. You may specify which device to use or pass in NULL to use any available device. If IPv6 is included in Nucleus NET, and you specify NULL as the **dv\_name**, the server processes both incoming IPv4 and IPv6 requests. However, if IPv6 is not included in Nucleus NET or you specify a valid parameter as the **dv\_name**, the server processes only incoming IPv4 requests.

### Example

```
CHAR    *dev_name;
STATUS  status;

/* Get the interface name of the network device. */
if (INCLUDE_IPV6 == NU_TRUE)
    dev_name = NU_NULL;
else
    /* Allocate memory to dev_name to hold the IPv4/IPv6 address. */
    /* Parse device structure and get device name. */

Status = TFTP_Server_Init(dev_name);

if (status != NU_SUCCESS)
    /* Error initializing TFTP Server */
```

### Related Topics

[TFTP Server Functions](#)

## TFTP\_Server\_Uninit

This function uninitializes the TFTP Server.

### Usage

```
STATUS TFTP_Server_Uninit (VOID);
```

### Return Values

- **NU\_SUCCESS**  
Upon successful completion.
- Otherwise, the routine returns the following Nucleus XPROT error code:
  - **NU\_INVALID**  
The TFTP Server failed to stop. Possible causes could be errors freeing tasks or memory.

### Description

This is the external interface to stop the TFTP service within your application. This routine waits for current clients to complete execution. All resources allocated to the TFTP Server application are deallocated. The server can be restarted with the TFTP\_Server\_Init function.

### Example

```
CHAR    *dev_name;
STATUS  status;

/* Allocate memory to dev_name to hold the IPv4/IPv6 address. */
/* Get the interface name of the network device. */
. . .

/* Start the TFTP Server. */
Status = TFTP_Server_Init(dev_name);

if (status != NU_SUCCESS)
    /* Error initializing TFTP Server */

/* Wait for five minutes to service TFTP clients. */
NU_Sleep(5 * 60 * TICKS_PER_SECOND);

/* Uninitialize TFTP Service after five minutes of startup */
status = TFTP_Server_Uninit();

if (status != NU_SUCCESS)
    /* Error uninitializing TFTP Server */
```

## Related Topics

[TFTP Server Functions](#)

# TFTP Server Shell Commands.

This section describes the TFTP shell command:

- [tftps](#)

## tftps

TFTP Server control and information. This command allows for basic control of the TFTP server on the target and allows for discovery of basic information about the TFTP server (such as the current directory path).

### Usage

```
tftps <operation>
```

### Arguments

- Mandatory
  - <operation>
  - dir - Show working directory for the TFTP server.
  - start - Start the TFTP server.
  - stop - Stop the TFTP server.

### Return Values

- If operation successful:
  - dir - show TFTP Server directory path.
  - start - show message that TFTP Server is started.
  - stop - show message that TFTP Server is stopped.
- If operation fails
  - Applicable error string.

### Examples

```
> tftps
ERROR: Invalid Usage!
Format: tftps <operation>
Where: operation = dir, start, or stop

>tftps dir
TFTP Server Directory = A:\

> tftps stop
Shutting down TFTP Server...
TFTP Server Shut Down Complete!

> tftps stop
ERROR: TFTP Server Already Shut Down!

> tftps start
Starting TFTP Server...
TFTP Server Running
```



## Related Topics

[NET Shell Commands](#)

[TFTP Server Shell Commands.](#)

# NET Notification Module

Nucleus NET versions 5.2 and above provide a notification module which can be used by the application to receive messages from lower layers and also receive advanced debugging information.

## NET Notification Module Data Structures

This section defines the data structures used within the NET Notification Module.

### NET\_NTIFY\_Debug\_Struct

NET\_NTIFY\_Debug\_Struct is the typedef name for the NET\_NTIFY\_DEBUG\_STRUCT data structure.

```
typedef struct NET_NTIFY_DEBUG_STRUCT
{
    INT    net_ntfy_type;
    VOID   *net_ntfy_info;
    INT    net_ntfy_length;
} NET_NTIFY_Debug_Struct;
```

The members of the structure are defined in Table 10-13.

**Table 10-13. NET\_NTIFY\_Debug\_Struct**

Member	Description
net_ntfy_type	The event-specific event code.
net_ntfy_info	A pointer to event-specific data.
net_ntfy_length	The length of the event-specific data.

## Related Topics

[NET Notification Module Data Structures](#)

[Routine for sending Informational Messages to the Application](#)

### NET\_NTIFY\_STRUCT

NET\_NTIFY\_STRUCT is the typedef name for the NET\_Ntfy\_Struct data structure.  
NET\_NTIFY\_STRUCT is defined as follows:

```
typedef struct NET_Ntfy_Struct
```

```

{
    STATUS net_ntfy_status;
    union
    {
        NET_DEBUG_BUFFER_INFO_STRUCT net_ntfy_buffer_info;
        DV_DEVICE_ENTRY               net_ntfy_dev_info;
    } net_ntfy_info;

    CHAR    net_ntfy_file[NLOG_MAX_BUFFER_SIZE];
    INT     net_ntfy_line;
    NU_TASK *net_ntfy_task;
    INT     net_ntfy_type;
} NET_NTIFY_STRUCT;

```

The members of the structure are defined in Table 10-14.

**Table 10-14. NET\_NTIFY\_STRUCT**

Member	Description
net_ntfy_status	The Nucleus NET or Nucleus PLUS error code describing the reason for the notification.
net_ntfy_info	A union of data specific to the notification type. If net_ntfy_type is NET_NTIFY_ETHERNET, the DV_DEVICE_ENTRY parameter of this union is entered. Otherwise, the NET_DEBUG_BUFFER_INFO_STRUCT parameter of this union is entered.
net_ntfy_file	The file from which the notification routine was called.
net_ntfy_line	The line number which caused the notification routine to be called.
net_ntfy_task	Pointer to the task from which the notification routine was called or OS_NULL if called from within a LISR.
net_ntfy_type	The notification type. Notifications from Ethernet drivers have a value of NET_NTIFY_ETHERNET and general notifications have a value of NET_NTIFY_GENERAL.

## Related Topics

[NET Notification Module Data Structures](#)

[Debugging Module for Receiving Messages from the Application](#)

# Routine for sending Informational Messages to the Application

It is sometimes necessary for the application to be informed of certain occurrences at lower layers. For example, the application may want to be notified if the FTP Server has been stopped.

The routine `NET_Notify()` is used to pass messages to the application layer:

```
VOID NET_Notify (INT32          ntfy_event,
                CHAR          *file,
                INT           line,
                NU_TASK       *task_ptr,
                NET_NTIFY_Debug_Struct *debug_info);
```

Where:

- `ntfy_event`

The event about which to notify the application.

- `file`

Pointer to `__FILE__`

- `line`

`__LINE__`

- `task_ptr`

Pointer to the task being executed or `NU_NULL` if the task is not thread-safe.

- `debug_info`

A pointer to the information structure containing event-specific information.

Data structure `NET_NTIFY_Debug_Struct` is defined in the [NET Notification Module Data Structures](#) of this chapter.

## Debugging Module for Receiving Messages from the Application

Nucleus NET provides an advanced debugging module. Nucleus FTP Server and TFTP Server use this debugging module to pass critical error conditions that might happen while starting or stopping the respective service. The debug module must be enabled in `os/include/networking/net_cfg.h` by setting `NU_DEBUG_NET` to `NU_TRUE`. It is disabled by default.

The debug module sends a message of type `NET_NTIFY_STRUCT` to the queue `NET_NTIFY_Msg_Queue` for the critical events shown in [Table 10-15](#), occurring in Nucleus XPROT:

**Table 10-15. Debug Messages**

Reason	Value
Error allocating memory for Control Task structure.	<code>FTPS_CTASK_STRU_MALLOC_FAILURE</code>

**Table 10-15. Debug Messages (cont.)**

Reason	Value
Error allocating memory for Control Task.	FTPS_CTASK_MALLOC_FAILURE
Error creating the Control Task.	FTPS_CTASK_CREATE_FAILURE
Error binding Control Task to Kernel.	FTPS_CTASK_KERNEL_BIND_FAILURE
FTP Server has been terminated.	FTPS_SERVER_TERMINATED
TFTP Server has been terminated.	TFTPS_SERVER_TERMINATED

Data structure [NET\\_NTFY\\_STRUCT](#) is defined in the [NET Notification Module Data Structures](#) of this chapter.

## Example

Detect if the FTP Server has terminated due to a critical error:

```

STATUS status;
UNSIGNED actual_size;
NET_NTFY_Debug_Struct ntfy_struct;
NET_NTFY_STRUCT queue_message;

/* Initialize and start the FTP Server task. */
. . .

for(;;)
{
    memset(&queue_message, 0, sizeof(NET_NTFY_Debug_Struct));
    status = NU_Receive_From_Queue(&NET_NTFY_Msg_Queue,
                                   &queue_message,
                                   sizeof(NET_NTFY_Debug_Struct),
                                   &actual_size, NU_SUSPEND);

    if (status == NU_SUCCESS)
    {
        if (queue_message == FTPS_SERVER_TERMINATED)
        {
            /* A critical event has occurred. */
            /* FTP Server has been terminated.*/
        }
    }
}

```

See the [NET](#) chapter for more information on the Nucleus NET Notification Module.





























































































































































































































## Nucleus ReadyStart HTTP Lite Overview

HTTP Lite is a simple client/server implementation in ReadyStart that supports the GET, PUT, POST, and DELETE commands. This Nucleus HTTP Lite Server is intended for micro controllers with minimal functionality and a flat file system. This implementation complies with RFC 2616 (HTTP/1.1) and maintains a small footprint, suited for embedded environments.

### Note



HTTP Lite and the Nucleus WebServer cannot be enabled at the same time.

---

## Protocol Overview

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for data transfer across the Internet. The standard has grown from raw data, then adding metadata, and this iteration (RFC2616) adds an open-ended set of methods and headers that indicate the purpose of a request.

The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI (Uniform Resource Identifier), and protocol version, followed by a MIME (Multipurpose Internet Mail Extension)-like message containing request modifiers, client information, and, optionally, body content over a connection with a server. The server responds with a status line, including the version of the message protocol and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possibly entity-body content.

The command sets are implemented on the client-side and server-side of the protocol, as a device can act as either when communicating with other nodes.

From the client-side, these commands are handled by filling out a message, sending the request, and awaiting a response from the server.

From the server-side, a request is received, and the server then replies with data and/or the appropriate response.

## Coverage and Limitations

Anything outside the scope of functionality discussed within this document is not supported.

## Related Topics

[Related Topics](#)

[User Configuration Options](#)

# Directory Structure

The source code for the client and server is stored at:  
<install\_path>\readystart\nucleus\os\networking\http.

The header files are stored at: <install\_path>\readystart\nucleus\os\include\networking.

## Related Topics

[User Configuration Options](#)

[HTTP Lite Functionality](#)

# User Configuration Options

You can configure the following options in the Nucleus Configuration Editor (UI editor):

- **server\_enable**  
Toggles compile-time support for enabling the HTTP Lite Server.
- **client\_enable**  
Toggles compile-time support for enabling the HTTP Lite Client.
- **svr\_stack\_size**  
Size, in bytes, of the HTTP stack.
- **svr\_backlog**  
The maximum number of waiting connections to allow on the HTTP Server. The HTTP Server processes one connection at a time. If there are more connections in the queue than permitted by the backlog value, those connection attempts are “refused”.
- **timeout**  
The time, in seconds, to wait before timing out an inactive HTTP connection on either the client- or server-side.
- **srv\_rcv\_size**  
The maximum packet size that the HTTP Lite Server can receive in one call to the networking stack. Note that this setting must be at least as large as the entire HTTP header. The data portion can be received with successive calls to the networking stack.
- **cli\_rcv\_size**  
The maximum packet size that the client can receive in one call to the networking stack.



- `svr_default_mime`

Sets the default MIME type. If a URI does not specify a file extension, the URI uses this setting. By default, this is set to 0 which maps to plain text. For other Mime types, refer to the `HTTP_Mime_Table`. You can find the `HTTP_Mime_Table` in the file:

*`http_lite_svr.c`*

- `svr_default_uri`

Sets the default URI. The HTTP Lite Server uses this string as the target URI in incoming requests that do not specify a complete URI.

- `svr_token_heap`

The maximum number of tokens to parse and store in the index array for a POST command.

- `svr_uri_len`

The maximum acceptable length of an incoming URI within a client request.

- `include_put_file`

Toggles compile-time support for enabling files to be written to the local flat file system on the server with a PUT command when no plug-in is found for the incoming URI.

- `include_delete_file`

Toggle compile-time support for enabling files to be deleted from the local flat file system on the server with a DELETE command when no plug-in is found for the incoming URI.

- `include_get_file`

Toggle compile-time support for enabling files to be retrieved from the local flat file system on the server with a GET command when no plug-in is found for the incoming URI.

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

## Related Topics

[HTTP Lite Functionality](#)

# HTTP Lite Functionality

Both the client-side and the server-side implementations support the following commands:

- GET

You can find a definition of the GET command in section 9.3 of RFC 2616.

- POST

You can find a definition of the POST command in section 9.5 of RFC 2616.

- PUT

You can find a definition of the PUT command in section 9.6 of RFC 2616. Section 9.6 also describes the fundamental difference between the POST and PUT commands.

- DELETE

You can find a description of the DELETE command in section 9.7 of RFC 2616.

Both the client and the server block until data is received from the other side of the connection or until the “timeout” (described in [User Configuration Options](#)) expires.

Both the client and the server in the HTTP Lite implementation use the flat memory system and are partially RFC 2616 compliant with no backward compatibility for older HTTP versions.

The HTTP Lite Server services one connection at a time (as it is single-threaded), and provides an API to register plug-ins at run-time that are invoked via the respective registered URI.

## Related Topics

[User Configuration Options](#)

# Plug-ins Parsing Data and Responding to Clients

Plug-ins are registered from the API using [NU\\_HTTP\\_Lite\\_Register\\_Plugin](#) and removed using [NU\\_HTTP\\_Lite\\_Remove\\_Plugin](#).

The format of a plug-in is:

```
STATUS plug_in_function_name (INT method,  
                              CHAR *uri,  
                              HTTP_SVR_SESSIONS_STRUCT *http_ptr);
```

## Arguments

- method  
Method that invoked the plug-in.
- uri  
URI used to invoke the plug-in.
- http\_ptr  
HTTP server structure that contains the information pertinent to the call.

The elements of [Related Topics](#) related to the plug-in functionality are in the [HTTP Lite Data Structures](#) section of this chapter.

## Plug-in Usage

Plug-ins are invoked in the context of the HTTP Lite Server thread and therefore have access to all internal HTTP Lite Server internal routines and data structures. A plug-in should not use API routines because these routines obtain the semaphore, and the plug-in already has the semaphore.

The plug-in should use the following routines for the respective operations:

- `HTTP_Lite_Send_Status_Message`  
Send a status message to the client.
- `HTTP_Lite_Svr_Send`  
Send data to the client using chunks.
- `HTTP_Lite_Read_Buffer`  
Allocate a new buffer for the data and read data into it.

### Note



Refer to *http\_lite\_strg.c* for more detail about the default plug-ins provided to GET, PUT and DELETE data to a file in the flat file system; `HTTP_Lite_Svr_Get`, `HTTP_Lite_Svr_Put`, and `HTTP_Lite_Svr_Delete`.

---

## Example of Parsing Data From a POST Command

This example enables or disables the default command for a PUT, GET, or DELETE operation.

```
STATUS http_lite_toggle_default_plugin(INT                type,
                                      CHAR                *uri,
                                      HTTP_SVR_SESSION_STRUCT *http_ptr)
{
    INT    method, operation, i, index;
    STATUS status;

    /* Parse out the method and operation.  This routine can process
     * only a single command with each POST operation.
     */
    for (i = 0; http_ptr->token_list.token_array[i] != -1; i++)
    {
        index = http_ptr->token_list.token_array[i];

        /* If the value is not 0 or 1, this must be the method. */
        if (NU_ATOI(&http_ptr->token_list.token_string[index +
            strlen(&http_ptr->token_list.token_string[index]) + 1]) > 1)
```

```
        method = NU_ATOI(&http_ptr->token_list.token_string[index
                        + strlen(&http_ptr->token_list.token_string[index]
                        + 1));

    /* Otherwise, this is the operation. */
    else

        operation = NU_ATOI(&http_ptr->token_list.token_string[index+
                        strlen(&http_ptr->token_list.token_string[index]
                        + 1));
    }

    /* Enable the default plug-in for the method. */
    if (operation == 1)
    {
        switch (method)
        {
        case HTTP_GET:
            status = HTTP_Lite_Register_Plugin(HTTP_Lite_Svr_Get,
                                                NU_NULL, HTTP_LITE_GET);

            break;

        case HTTP_PUT:
            status = HTTP_Lite_Register_Plugin(HTTP_Lite_Svr_Put,
                                                NU_NULL, HTTP_LITE_PUT);

            break;

        case HTTP_DELETE:
            status = HTTP_Lite_Register_Plugin(HTTP_Lite_Svr_Delete,
                                                NU_NULL, HTTP_LITE_DELETE);

            break;

        default:
            break;
        }
    }

    /* Disable the default plug-in for the method. */
    else
    {
        switch (method)
        {
        case HTTP_GET:
            HTTP_Lite_Remove_Plugin(NU_NULL, HTTP_LITE_GET);

            break;

        case HTTP_PUT:
            HTTP_Lite_Remove_Plugin(NU_NULL, HTTP_LITE_PUT);

            break;

        case HTTP_DELETE:
            HTTP_Lite_Remove_Plugin(NU_NULL, HTTP_LITE_DELETE);

            break;

        default:
            break;
        }
    }
}
```

```
status = HTTP_Lite_Send_Status_Message(HTTP_PROTO_OK, HTTP_OK);

return (status);
}
```

## Related Topics

[HTTP Lite Functionality](#)

[Plug-ins Parsing Data and Responding to Clients](#)

# HTTP Lite Data Structures

This section provides details on data structures defined for usage with HTTP Lite specific APIs.

- [NU\\_HTTP\\_CLIENT](#)
- [NU\\_HTTP\\_SSL\\_STRUCT](#)
- [HTTP\\_SVR\\_SESSION\\_STRUCT](#)
- [HTTP\\_TOKEN\\_INFO](#)

## NU\_HTTP\_CLIENT

NU\_HTTP\_CLIENT is the typedef name for the \_nu\_http\_client data structure.

```
typedef struct _nu_http_client
{
    CHAR            *uri;
    CHAR            *pdata;
    CHAR            *ptype;
    NU\_HTTP\_SSL\_STRUCT ssl_struct;
    UINT32          plength;
    UINT16          type_size;
    UINT8           overwrite;
    UINT8           padN[1];
} NU_HTTP_CLIENT;
```

The members of the structure are defined in [Table 11-1](#).

**Table 11-1. NU\_HTTP\_CLIENT**

Member	Description
uri	The uri for the respective operation. If the operation is to be performed over SSL, the application uses https in the URI name instead of http.
pdata	Pointer to the buffer for the respective data.
ptype	Pointer to the type of data.

**Table 11-1. NU\_HTTP\_CLIENT (cont.)**

Member	Description
ssl_struct	<a href="#">NU_HTTP_SSL_STRUCT</a> data structure used when connecting over SSL.
plength	The length of the buffer for the respective data.
type_size	The size of the pointer ptype.
overwrite	NU_TRUE or NU_FALSE to overwrite an existing resource.

## Related Topics

[HTTP Lite Data Structures](#)

[NU\\_HTTP\\_Lite\\_Client\\_Get](#)

[NU\\_HTTP\\_Lite\\_Client\\_Post](#)

[NU\\_HTTP\\_Lite\\_Client\\_Put](#)

[NU\\_HTTP\\_Lite\\_Client\\_Delete](#)

[NU\\_HTTP\\_SSL\\_STRUCT](#)

## NU\_HTTP\_SSL\_STRUCT

NU\_HTTP\_SSL\_STRUCT is the typedef name for the `_nu_http_ssl_struct` data structure used if the connection is made over SSL.

```
typedef struct _nu_http_ssl_struct
{
    CHAR    *ssl_ca;
    CHAR    *ssl_cert;
    CHAR    *ssl_key;
    INT     ssl_ca_size;
    INT     ssl_cert_size;
    INT     ssl_key_size;
    INT     ssl_flags;
} NU_HTTP_SSL_STRUCT;
```

The members of the structure are defined in [Table 11-2](#):

**Table 11-2. NU\_HTTP\_SSL\_STRUCT**

Member	Description
ssl_ca	Pointer to the SSL CA to use for the client connection.
ssl_cert	Pointer to the certificate to send to the server if requested.
ssl_key	Pointer to the key to send to the server if requested.
ssl_ca_size	The size of the SSL CA if not being retrieved from a file system.

**Table 11-2. NU\_HTTP\_SSL\_STRUCT (cont.)**

Member	Description
ssl_cert_size	The size of the SSL certificate if not being retrieved from a file system.
ssl_key_size	The size of the SSL key if not being retrieved from a file system.
ssl_flags	<p>ssl_flags are as follows:</p> <ul style="list-style-type: none"> <li>• <b>NU_HTTP_CERT_PEM_PTR</b> The certificate points to a PEM certificate in volatile memory.</li> <li>• <b>NU_HTTP_CERT_DER_PTR</b> The certificate points to a binary certificate in volatile memory.</li> <li>• <b>NU_HTTP_CERT_FILE</b> The certificate is stored on a local drive.</li> <li>• <b>NU_HTTP_KEY_FILE</b> The key sent to the foreign server is stored on a local drive.</li> <li>• <b>NU_HTTP_KEY_PEM_PTR</b> The key sent to the foreign server points to a PEM file in volatile memory.</li> <li>• <b>NU_HTTP_KEY_DER_PTR</b> The key sent to the foreign server points to a binary certificate in volatile memory.</li> <li>• <b>NU_HTTP_CA_PEM_PTR</b> The CA list points to a PEM certificate in volatile memory.</li> <li>• <b>NU_HTTP_CA_DER_PTR</b> The CA list points to a binary certificate in volatile memory.</li> <li>• <b>NU_HTTP_CA_FILE</b> The CA list is stored on a local drive.</li> <li>• <b>NU_HTTP_VERIFY_PEER</b> Verify the foreign server certificate when connecting via a client connection.</li> <li>• <b>NU_HTTP_NO_DOMAIN_CHECK</b> Do not check the domain name before connecting over SSL.</li> </ul>

## Related Topics


[HTTP Lite Data Structures](#)

[NU\\_HTTP\\_CLIENT](#)

## HTTP\_SVR\_SESSION\_STRUCT

HTTP\_SVR\_SESSION\_STRUCT is the typedef name for the \_http\_svr\_session\_struct data structure.

---

**Note**  Only the header is received by the HTTP Server. The plug-in must issue subsequent calls to HTTP\_Lite\_Read\_Buffer() to receive the data portion of the packet.

---

```
typedef struct _http_svr_session_struct
{
    CHAR                *buffer;
    UINT32              in_hdr_size;
    INT                 socketd;
    HTTP_TOKEN_INFO     token_list;
    HTTP_TOKEN_INFO     token_query_list;
} HTTP_SVR_SESSION_STRUCT;
```

The members of the structure are defined in Table 11-3. Some members of this structure have been omitted as they are used internally.

**Table 11-3. HTTP\_SVR\_SESSION\_STRUCT**

Member	Description
buffer	Pointer to the incoming header.
in_hdr_size	The length of the data in the incoming header.
socketd	The socket over which data can be sent and received for this connection.
token_list	The parsed data for a POST command. <a href="#">HTTP_TOKEN_INFO</a> is detailed in the <a href="#">HTTP Lite Data Structures</a> section of this chapter.
token_query_list	The parsed token data found within the URI.

## Related Topics

[HTTP Lite Data Structures](#)

[HTTP\\_TOKEN\\_INFO](#)

[Plug-ins Parsing Data and Responding to Clients](#)

## HTTP\_TOKEN\_INFO

HTTP\_TOKEN\_INFO is the typedef name for the \_http\_token\_info data structure.

```
typedef struct _http_token_info
{
    INT16               token_array[HTTP_TOKEN_HEAP + 1];
    CHAR HUGE           *token_string;
    UINT32              token_string_len;
} HTTP_TOKEN_INFO;
```



The members of the structure are defined in Table 11-4. Some members of this structure have been omitted as they are used internally.

**Table 11-4. HTTP\_TOKEN\_INFO**

Member	Description
token_array	Each element of the array denotes the index into token_string, where the next parsed token can be found. The end of the tokens are denoted by a -1 value in the array.
token_string	Points to the null-terminated data being indexed by the token_array.
token_string_len	The total length of the data in the token_string.

## Related Topics

[HTTP Lite Data Structures](#)

[Related Topics](#)

[Plug-ins Parsing Data and Responding to Clients](#)

# HTTP Lite Function APIs

This section describes the APIs of the following functions:

- [NU\\_HTTP\\_Lite\\_Client\\_Get](#)
- [NU\\_HTTP\\_Lite\\_Client\\_Post](#)
- [NU\\_HTTP\\_Lite\\_Client\\_Put](#)
- [NU\\_HTTP\\_Lite\\_Client\\_Delete](#)
- [NU\\_HTTP\\_Lite\\_Client\\_Init](#)
- [NU\\_HTTP\\_Lite\\_Server\\_Init](#)
- [NU\\_HTTP\\_Lite\\_Configure\\_SSL](#)
- [NU\\_HTTP\\_Lite\\_Server\\_Shutdown](#)
- [NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#)
- [NU\\_HTTP\\_Lite\\_Delete\\_Session\\_Handle](#)
- [NU\\_HTTP\\_Lite\\_Register\\_Plugin](#)
- [NU\\_HTTP\\_Lite\\_Remove\\_Plugin](#)
- [NU\\_HTTP\\_Lite\\_Create\\_File](#)

- [NU\\_HTTP\\_Lite\\_Write\\_File](#)
- [NU\\_HTTP\\_Lite\\_Delete\\_File](#)
- [NU\\_HTTP\\_Lite\\_Register\\_Upgrade\\_Plugin](#)
- [NU\\_HTTP\\_Lite\\_Remove\\_Upgrade\\_Plugin](#)

---

**Note**



These functions return the codes described in Return Values. They also return the server return values as described in section 10 of RFC 2616.

---

## NU\_HTTP\_Lite\_Client\_Get

Executes the GET command for the specified URI.

### Usage

```
STATUS NU_HTTP_Lite_Client_Get (INT32          session_id,
                                NU_HTTP_CLIENT *cli_struct,
                                STATUS          *http_status);
```

### Arguments

- **session\_id**  
 The ID that identifies a specific session. This value is returned by a prior call to [NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#).
- **cli\_struct**  
 Pointer to the [NU\\_HTTP\\_CLIENT](#) structure. This structure is described in the [HTTP Lite Data Structures](#) section.
- **http\_status**  
 Pointer to the status returned by the remote server.

### Return Values

- **NU\_SUCCESS**  
 The operation completed successfully.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **HTTP\_INVALID\_URI**  
 The URI is not formed properly.
- **HTTP\_NO\_IP\_NODE**  
 The host name is invalid.
- **HTTP\_ERROR\_DATA\_WRITE**  
 Could not send data to the server.
- **HTTP\_INVALID\_HEADER\_READ**  
 No response from the server.
- **HTTP\_INVALID\_HEADER\_PARSE**  
 The server is not using a compatible HTTP version.
- **HTTP\_ERROR\_UNREAD\_DATA**  
 Some data was received but more data may be pending on the socket.

- HTTP\_SSL\_ERROR  
An error occurred while connecting via SSL.
- Operating-system specific error  
Indicates the nature of the failure: out of memory, could not connect to server, and so on.

## Example

```
STATUS          status, http_status;
INT32           session_id;
CHAR            uri[HTTP_URI_LEN] = {0};
CHAR            type[256] = {0};
NU_HTTP_CLIENT http_struct;

/* Create a valid URI. */
strcpy(uri, "http://mentor.com/");

/* Clear the NU_HTTP_CLIENT Structure. */
memset(&http_struct, 0, sizeof(NU_HTTP_CLIENT));

http_struct.uri = uri;
http_struct.pdata = NU_NULL;
http_struct.plength = 0; /* Use the internal default length. */
http_struct.ptype = type;
http_struct.type_size = 256;

/* Get a valid session ID. */
session_id = NU_HTTP_Lite_Create_Session_Handle();
if (session_id >= 0)
{
    /* Get the page. */
    status = NU_HTTP_Lite_Client_Get(session_id, &http_struct,
                                     &http_status);

    while (status == HTTP_ERROR_UNREAD_DATA)
    {
        /* Receive the rest of the data */
        NU_Deallocate_Memory(http_struct.pdata);
        http_struct.plength = 0;

        status = NU_HTTP_Lite_Client_Get(session_id, &http_struct,
                                         &http_status);

        . . .
    }
}
```

## Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Client\\_Post](#)

## NU\_HTTP\_Lite\_Client\_Post

Executes the POST command for the specified URI.

### Usage

```
STATUS NU_HTTP_Lite_Client_Post (INT32          session_id,
                                NU_HTTP_CLIENT *cli_struct,
                                STATUS          *http_status);
```

### Arguments

- **session\_id**  
 The ID that identifies a specific session. This value is returned by a prior call to [NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#).
- **cli\_struct**  
 Pointer to the [NU\\_HTTP\\_CLIENT](#) structure. This structure is described in the [HTTP Lite Data Structures](#) section.
- **http\_status**  
 Pointer to the status returned by the remote server.

### Return Values

- **NU\_SUCCESS**  
 The operation completed successfully.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **HTTP\_INVALID\_URI**  
 The URI is not formed properly.
- **HTTP\_NO\_IP\_NODE**  
 The host name is invalid.
- **HTTP\_ERROR\_DATA\_WRITE**  
 Could not send data to the server.
- **HTTP\_INVALID\_HEADER\_READ**  
 No response from the server.
- **HTTP\_INVALID\_HEADER\_PARSE**  
 The server is not using a compatible HTTP version.
- **HTTP\_SSL\_ERROR**  
 An error occurred while connecting over SSL.

- Operating-system specific error

Indicates the nature of the failure: out of memory, could not connect to server, and so on.

## Example

```
STATUS          status, http_status;
INT32           session_id;
CHAR            uri[HTTP_URI_LEN] = {0};
CHAR            type[256] = {0};
CHAR            data[] = "This text should be sent to the server.";
NU_HTTP_CLIENT http_struct;

/* Create the URI. */
strcpy(uri, "http://mentor.com/example_post");

/* Clear the NU_HTTP_CLIENT Structure. */
memset(&http_struct, 0, sizeof(NU_HTTP_CLIENT));

http_struct.uri = uri;
http_struct.pdata = data;
http_struct.plength = strlen(data);
http_struct.ptype = type;
http_struct.overwrite = NU_TRUE;

/* Get a valid session ID. */
session_id = NU_HTTP_Lite_Create_Session_Handle();
if (session_id >= 0)
{
    /* Make the Post call */
    status = NU_HTTP_Lite_Client_Post(session_id, &http_struct,
                                     &http_status);

    . . .
}
```

## Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Client\\_Get](#)

## NU\_HTTP\_Lite\_Client\_Put

Executes the PUT command for the specified URI.

### Usage

```
STATUS NU_HTTP_Lite_Client_Put (INT32      session_id,
                               NU_HTTP_CLIENT *cli_struct,
                               STATUS      *http_status);
```

### Arguments

- **session\_id**  
 The ID that identifies a specific session. This value is returned by a prior call to [NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#).
- **cli\_struct**  
 Pointer to a [NU\\_HTTP\\_CLIENT](#) structure. This structure is described in the [HTTP Lite Data Structures](#) section.
- **http\_status**  
 Pointer to the status returned by the remote server.

### Return Values

- **NU\_SUCCESS**  
 The operation completed successfully.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **HTTP\_INVALID\_URI**  
 The URI is not formed properly.
- **HTTP\_NO\_IP\_NODE**  
 The host name is invalid.
- **HTTP\_ERROR\_DATA\_WRITE**  
 Could not send data to the server.
- **HTTP\_INVALID\_HEADER\_READ**  
 No response from the server.
- **HTTP\_INVALID\_HEADER\_PARSE**  
 The server is not using a compatible HTTP version.
- **HTTP\_SSL\_ERROR**  
 An error occurred while connecting over SSL.

- Operating-system specific error

Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS          status, http_status;
INT32           session_id;
CHAR            uri[HTTP_URI_LEN] = {0};
CHAR            type[256] = {0};
CHAR            data[] = "This text should be sent to the server.";
NU_HTTP_CLIENT  http_struct;

/* Create the URI. */
strcpy(uri, "http://mentor.com/new_file.htm");

/* Clear the NU_HTTP_CLIENT Structure. */
memset(&http_struct, 0, sizeof(NU_HTTP_CLIENT));

http_struct.uri = uri;
http_struct.pdata = data;
http_struct.plength = strlen(data);
http_struct.ptype = type;
http_struct.overwrite = NU_TRUE;

/* Get a valid session ID. */
session_id = NU_HTTP_Lite_Create_Session_Handle();
if (session_id >= 0)
{
    /* Make the Put call */
    status = NU_HTTP_Lite_Client_Put(session_id, &http_struct,
                                     &http_status);

    . . .
}
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Client\\_Post](#)



## NU\_HTTP\_Lite\_Client\_Delete

Executes the DELETE command for the specified URI.

### Usage

```
STATUS NU_HTTP_Lite_Client_Delete(INT32      session_id,
                                  NU_HTTP_CLIENT *cli_struct,
                                  STATUS      *http_status);
```

### Arguments

- **session\_id**  
 The ID that identifies a specific session. This value is returned by a prior call to [NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#).
- **cli\_struct**  
 Pointer to a [NU\\_HTTP\\_CLIENT](#) structure. This structure is described in the [HTTP Lite Data Structures](#) section.
- **http\_status**  
 Pointer to the status returned by the remote server.

### Return Values

- **NU\_SUCCESS**  
 The operation completed successfully.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **HTTP\_INVALID\_URI**  
 The URI is not formed properly.
- **HTTP\_NO\_IP\_NODE**  
 The host name is invalid.
- **HTTP\_ERROR\_DATA\_WRITE**  
 Could not send data to the server.
- **HTTP\_INVALID\_HEADER\_READ**  
 No response from the server.
- **HTTP\_INVALID\_HEADER\_PARSE**  
 The server is not using a compatible HTTP version.
- **HTTP\_SSL\_ERROR**  
 An error occurred while connecting over SSL.

- Operating-system specific error

Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS          status, http_status;
INT32           session_id;
CHAR            uri[HTTP_URI_LEN] = {0};
CHAR            type[256] = {0};
CHAR            data[] = "This text should be sent to the server.";
NU_HTTP_CLIENT  http_struct;

/* Create the URI. */
strcpy(uri, "http://mentor.com/delete.htm");

/* Clear the NU_HTTP_CLIENT Structure. */
memset(&http_struct, 0, sizeof(NU_HTTP_CLIENT));

http_struct.uri = uri;

/* Get a valid session ID. */
session_id = NU_HTTP_Lite_Create_Session_Handle();
if (session_id >= 0)
{
    /* Make the Delete call. */
    status = NU_HTTP_Lite_Client_Delete(session_id, &http_struct,
                                         &http_status);

    . . .
}
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Client\\_Put](#)

## NU\_HTTP\_Lite\_Client\_Init

This function initializes all resources for the HTTP Client module.

### Usage

```
STATUS NU_HTTP_Lite_Client_Init(VOID);
```

### Arguments

- None

### Return Values

- **NU\_SUCCESS**  
Operation completed successfully.
- Operating-system specific error  
Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS status;  
  
/* Initialize the HTTP Lite Client. */  
status = NU_HTTP_Lite_Client_Init();
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Client\\_Delete](#)

## NU\_HTTP\_Lite\_Server\_Init

This function initializes all resources for the HTTP Lite Server module and starts the thread that listens for requests on the HTTP port. The server listens on port 80 by default, and port 443 if SSL is compiled into the system.

### Usage

```
STATUS NU_HTTP_Lite_Server_Init (VOID);
```

### Arguments

- None

### Return Values

- NU\_SUCCESS  
Operation completed successfully.
- Operating-system specific error  
Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS status;  
  
/* Initialize the HTTP Lite Server. */  
status = NU_HTTP_Lite_Server_Init();
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Server\\_Shutdown](#)

## NU\_HTTP\_Lite\_Configure\_SSL

This function configures the SSL certificate and key that is used by the HTTP Lite Server, and can change the certificate/key at run-time. This function must be invoked before the HTTP Lite Server can service any secure connections.

### Usage

```
status = NU_HTTP_Lite_Configure_SSL(CHAR    *cert,
                                   UINT8   cert_type,
                                   INT      cert_size,
                                   CHAR     *key,
                                   UINT8   key_type,
                                   INT      key_size,
                                   CHAR     *ca_list,
                                   UINT8   ca_type,
                                   INT      ca_size,
                                   INT      flag);
```

### Arguments

- **cert**  
 A pointer to the certificate to use for new incoming connections.
- **cert\_type**  
 The type of data pointed to by the cert pointer:  
     **HTTP\_CERT\_FILE**  
         cert points to a file. Binary files can be represented as either *.der* or *.cer*.  
     **HTTP\_CERT\_PEM\_PTR**  
         cert points to a buffer of data representing a certificate in non-binary format.  
     **HTTP\_CERT\_DER\_PTR** -  
         cert points to a buffer of data representing a certificate in binary format.
- **cert\_size**  
 The size of the buffer of data that holds the certificate. Used only if cert\_type is not **HTTP\_CERT\_FILE**.
- **key**  
 A pointer to the key to use for new incoming connections.
- **key\_type**  
 The type of data pointed to by the key pointer:  
     **HTTP\_KEY\_FILE**  
         key points to a file. Binary files can be represented as either *.der* or *.cer*.  
     **HTTP\_KEY\_PEM\_PTR**  
         key points to a buffer of data representing a key in non-binary format.

#### HTTP\_KEY\_DER\_PTR

key points to a buffer of data representing a key in binary format.

- key\_size

The size of the buffer of data that holds the key. Used only if key\_type is not HTTP\_KEY\_FILE.

- ca\_list
- A pointer to the Certificate Authority(ies) to use to validate the client.
- ca\_type

The type of data in the ca\_list pointer:

#### HTTP\_CA\_FILE

ca\_list points to either a binary or PEM file.

#### HTTP\_CA\_PEM\_PTR

ca\_list points to a buffer of data representing a CA in PEM format.

#### HTTP\_CA\_DER\_PTR

ca\_list points to a buffer of data representing a CA in binary format.

- ca\_size

Used only if ca\_type is not HTTP\_CA\_FILE. The size of the buffer of data that holds the CA.

- flags

Flags used to indicate other desired behavior:

#### NU\_HTTP\_LITE\_SVR\_VERIFY\_CLIENT

Send a certificate request to the client and check the certificate.

#### NU\_HTTP\_LITE\_SVR\_VERIFY\_FAIL

Fail the connection if the client fails to send a certificate when requested to do so.

### Return Values

- NU\_SUCCESS

Function completed successfully, SSL was configured.

- NU\_INVALID\_PARM

One of the input parameters is invalid.

- HTTP\_SSL\_ERROR

An error occurred when trying to configure the certificate or key.

- An operating-system specific error.

## Examples

```
STATUS status;
CHAR cert[] = "B:\\cert.pem";
CHAR key[] = "B:\\key.pem";
CHAR ca[] = "B:\\ca-list.pem";

status = NU_HTTP_Lite_Configure_SSL(cert, HTTP_CERT_FILE, 0, key,
                                   HTTP_KEY_FILE, 0, ca, HTTP_CA_FILE,
                                   0, NU_HTTP_LITE_SVR_VERIFY_CLIENT);
```

## Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#)

## NU\_HTTP\_Lite\_Server\_Shutdown

Stops the HTTP Lite Server thread and deallocates all resources associated with the server.

### Usage

```
VOID NU_HTTP_Lite_Server_Shutdown(VOID);
```

### Arguments

- None

### Return Values

- None

### Example

```
STATUS status;

/* Initialize the HTTP Lite Server. */
status = NU_HTTP_Lite_Server_Init();

. . .

/* Shut the HTTP Lite Server down. */
NU_HTTP_Lite_Server_Shutdown();
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Server\\_Init](#)



## NU\_HTTP\_Lite\_Create\_Session\_Handle

This function creates a unique session handle used for communicating with a remote server.

### Usage

```
INT32 NU_HTTP_Lite_Create_Session_Handle(VOID);
```

### Arguments

- None

### Return Values

- 32-bit signed session handle ID
- Operating-system specific error code

Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
INT32 session_id;  
  
/* Open a new session handle. */  
session_id = NU_HTTP_Lite_Create_Session_Handle();
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Delete\\_Session\\_Handle](#)

## NU\_HTTP\_Lite\_Delete\_Session\_Handle

This function deletes a session handle and deallocates all resources associated with the session handle.

### Usage

```
STATUS NU_HTTP_Lite_Delete_Session_Handle(INT32 session_id);
```

### Arguments

- `session_id`  
The session handle ID to delete.

### Return Values

- `NU_SUCCESS`  
Operation completed successfully.
- Operating-system specific error  
Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
INT32 session_id;  
STATUS status;  
  
/* Open a new session handle. */  
session_id = NU_HTTP_Lite_Create_Session_Handle();  
  
. . .  
  
/* Delete the session handle. */  
status = NU_HTTP_Lite_Delete_Session_Handle(session_id);
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Create\\_Session\\_Handle](#)

## NU\_HTTP\_Lite\_Register\_Plugin

This function registers a plug-in on the local server that can be invoked by a remote client.

### Usage

```
STATUS NU_HTTP_Lite_Register_Plugin (
    STATUS (*plug_in)(INT, CHAR*, HTTP_SVR_SESSION_STRUCT*),
    CHAR    *uri,
    INT     methods);
```

### Arguments

- **plug\_in**  
 Pointer to the routine that is invoked when a remote client specifies the URI associated with this plug-in.
- **uri**  
 Pointer to the unique URI a remote client uses to invoke this plug-in.
- **methods**  
 The methods for which this plug-in is allowed to be invoked:

```
HTTP_LITE_PUT
HTTP_LITE_POST
HTTP_LITE_DELETE
HTTP_LITE_GET
```

### Return Values

- **NU\_SUCCESS**  
 The plug-in was registered.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **Operating-system specific error**  
 Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS status;

/* Register a new plug-in that is accessed via the URI
 * "do_something" through a POST or PUT command. */
status = NU_HTTP_Lite_Register_Plugin(do_something_function,
                                     "do_something",
                                     HTTP_LITE_POST | HTTP_LITE_PUT);
```

## Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Remove\\_Plugin](#)

## NU\_HTTP\_Lite\_Remove\_Plugin

This function removes a plug-in from the system and deallocates all resources associated with the plug-in.

### Usage

```
STATUS NU_HTTP_Lite_Remove_Plugin(CHAR *uri,  
                                  INT  methods);
```

### Arguments

- uri  
Pointer to the URI associated with the plug-in to remove.
- methods  
The methods for which this plug-in is to be removed:

```
HTTP_LITE_PUT  
HTTP_LITE_POST  
HTTP_LITE_DELETE  
HTTP_LITE_GET
```

### Return Values

- NU\_SUCCESS  
The plug-in was removed.
- Operating-system specific error  
Indicates the nature of the failure: out of memory, could not connect to server, and so on.

### Example

```
STATUS status;  
  
/* Register a new plug-in that is accessed via the URI "do_something"  
 * for a POST or PUT command.  
 */  
status = NU_HTTP_Lite_Register_Plugin( do_something_function,  
                                       "do_something",  
                                       HTTP_LITE_POST | HTTP_LITE_PUT);  
  
. . .  
  
/* Remove the plug-in for the PUT method. */  
status = NU_HTTP_Lite_Remove_Plugin("do_something", HTTP_LITE_PUT);
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Register\\_Plugin](#)

## NU\_HTTP\_Lite\_Create\_File

This function creates a file on the server. The HTTP Lite Server uses a linked-list to store files in RAM. There is no permanent storage mechanism.

### Usage

```
STATUS NU_HTTP_Lite_Create_File (CHAR *fname);
```

### Arguments

- fname  
Pointer to the name of the file to create.

### Return Values

- NU\_SUCCESS  
The file was created successfully.
- HTTP\_INVALID\_PARAMETER  
The input is invalid.
- HTTP\_FILE\_ALREADY\_EXISTS  
The filename already exists.
- Operating-system specific error  
Indicates the nature of the failure: could not obtain semaphore, out of memory, and so on.

### Example

```
STATUS status;  
  
/* Create a file that will contain 100 bytes of data. */  
status = NU_HTTP_Lite_Create_File("new_file.txt");
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Delete\\_File](#)

## NU\_HTTP\_Lite\_Write\_File

This function writes data to the end of an existing file.

### Usage

```
STATUS NU_HTTP_Lite_Write_File (CHAR    *fname,
                                UINT32  len,
                                CHAR    *buffer);
```

### Arguments

- **fname**  
 Pointer to the name of the file to which to write the data.
- **len**  
 Length of the data being written into the file.
- **buffer**  
 Pointer to the data to write.

### Return Values

- **NU\_SUCCESS**  
 The file was written to successfully.
- **HTTP\_INVALID\_PARAMETER**  
 The input is invalid.
- **Operating-system specific error**  
 Indicates the nature of the failure: could not obtain semaphore, out of memory, and so on.

### Example

```
STATUS status;
CHAR    buffer[10];

/* Create a file that will contain 100 bytes of data. */
status = NU_HTTP_Lite_Create_File("new_file.txt", 100);

/* Write 10 bytes of data to the file. */
status = NU_HTTP_Lite_Write_File("new_file.txt", 10, buffer);
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Create\\_File](#)

## NU\_HTTP\_Lite\_Delete\_File

This function deletes a file from the server.

### Usage

```
STATUS NU_HTTP_Lite_Delete_File(CHAR *fname);
```

### Arguments

- **fname**  
Pointer to the name of the file to delete.

### Return Values

- **NU\_SUCCESS**  
The file was deleted successfully.
- **HTTP\_INVALID\_PARAMETER**  
The input is invalid.
- **Operating-system specific error**  
Indicates the nature of the failure: could not obtain semaphore, and so on.

### Example

```
STATUS status;  
  
/* Create a file that will contain 100 bytes of data. */  
status = NU_HTTP_Lite_Create_File("new_file.txt", 100);  
  
. . .  
  
/* Delete the file. */  
status = NU_HTTP_Lite_Delete_File("new_file.txt");
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Create\\_File](#)



## NU\_HTTP\_Lite\_Register\_Upgrade\_Plugin

This function registers a plug-in for the specified upgrade request and method(s).

### Usage

```
STATUS NU_HTTP_Lite_Register_Upgrade_Plugin(
    STATUS (*plug_in)(INT, CHAR *, CHAR* INT, INT, VOID*),
    CHAR    *upgrade,
    INT     methods);
```

### Arguments

- **plug\_in**  
 The name of the plug-in function to invoke when an upgrade request is received for the specified method(s).
- **upgrade**  
 The string used to identify the plug-in. If NULL, the plug-in should be used as the default plug-in for the method(s) specified when any upgrade request is received that does not match a specific plug-in.
- **methods**  
 The method(s) for which the plug-in is allowed:

```
HTTP_LITE_PUT
HTTP_LITE_POST
HTTP_LITE_DELETE
HTTP_LITE_GET
```

### Return Values

- **NU\_SUCCESS**  
 The function completed successfully, the plug-in was registered.
- **HTTP\_INVALID\_PARAMETER**  
 One of the input parameters is invalid.
- **Operating-system specific error.**

### Examples

```
STATUS status;
/* Set up the HTTP upgrade on the server to the default routine. */
status =
NU_HTTP_Lite_Register_Upgrade_Plugin(NU_WSOX_Process_Upgrade_Request,
                                     "websocket", HTTP_LITE_GET);
```

### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Remove\\_Upgrade\\_Plugin](#)

## NU\_HTTP\_Lite\_Remove\_Upgrade\_Plugin

This function removes an upgrade plug-in.

### Usage

```
STATUS NU_HTTP_Lite_Remove_Upgrade_Plugin (CHAR *upgrade,  
                                           INT  methods);
```

### Arguments

- upgrade  
The string used to identify the plug-in.
- methods  
The method(s) for which the plug-in is deleted:

```
HTTP_LITE_PUT  
HTTP_LITE_POST  
HTTP_LITE_DELETE  
HTTP_LITE_GET
```

### Return Values

- NU\_SUCCESS  
Function completed successfully, the plug-in was removed. This routine returns NU\_SUCCESS whether the upgrade was found or not.

### Examples

```
STATUSs    status;  
  
/* Stop accepting new WebSocket connections from the HTTP server. */  
status =  
NU_HTTP_Lite_Remove_Upgrade_Plugin (NU_WSOX_Process_Upgrade_Request,  
                                     HTTP_LITE_GET);
```


### Related Topics

[HTTP Lite Function APIs](#)

[NU\\_HTTP\\_Lite\\_Register\\_Upgrade\\_Plugin](#)

Nucleus WebServ is a HyperText Transfer Protocol (HTTP) server designed for embedded environments. It maintains a very small memory footprint, while providing an extensive set of functions. Nucleus WebServ is designed to serve multi-media files to the World Wide Web (www) clients (for example, Web browsers) and to enable the remote control and configuration of the embedded platform where it is deployed.

---

 **Warning** In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## Web Server Overview

An HTTP web server communicates to clients over a TCP/IP connection. The most common medium for this connection is the Internet. The client sends requests to the server for a specific action to be performed. This action may be a simple GET that returns a web page or could be as complicated as commanding the server to reconfigure the system where it is located. Nucleus WebServ is designed to handle simple requests, and plug-ins can be written to further expand the area of usefulness of Nucleus WebServ so that more complicated tasks can be accomplished.

### Related Topics

[Hyper Text Transfer Protocol](#)

[Hyper Text Mark-up Language](#)

## Hyper Text Transfer Protocol

Hyper Text Transfer Protocol (HTTP) is a set of standards developed to ease communications across the Internet. It sets up a series of basic commands that servers must be able to perform. This protocol also specifies the terms that must be used to communicate between two entities on the Internet. The most current release of this protocol is 1.1.

The commands within HTTP that Nucleus WebServ supports are GET, HEAD, and POST. The GET command is used most often. This command is a request by the client to receive a file. This file can be in any form, HTML web page, multimedia files, Java applets, and so on. These files are immediately sent to the client by the server upon receiving the request. The response is

made up of an HTTP packet header, which describes the data being sent, followed by the data itself.

A specific type of GET request is the Server Side Include (SSI). The client sends a request to the Web Server in the form of a URL. The Web Server detects the *.ssi* extension on the request and then searches the requested file for the unique “<#” comment string. When found, the name associated with the special symbol is passed to the server along with any parameters that might be necessary. Once the action is complete, the result is returned to the client. What is returned depends on the SSI, it could take the form of an HTML page, a redirection of the client, or some other data file.

The HEAD command is used to determine if a file that is to be requested is too large for the client’s network to handle. When the server gets this request, it creates a packet just as it would for a GET request for the same file, with one exception, no data is attached to the packet. Once the entire request packet is completed it is sent to the client. This allows the client to verify the information about the data being sent will not cause problems within the client’s network.

The POST operation adds great flexibility to Nucleus WebServ. The POST operation allows the client to send information with its request. This may be the information gathered from an online form or the attachment of a file. How the server handles the post depends on the operation required.

The POST operations and SSI requests are implemented through plug-ins located on the server. These plug-ins are tailored to specifically respond to requests made by clients.

## Related Topics

[Web Server Overview](#)

[Hyper Text Mark-up Language](#)

## Hyper Text Mark-up Language

HTML is a common form of communication between a browser and server through an Internet connection. The flexibility of HTML allows for a server to answer any client request and, when necessary, easily create a web page dynamically. Today, software can be used to create web pages, thus abstracting the HTML layer from the creator. Documentation on the Internet can be found for a further and more in-depth explanation of HTML.

## Related Topics

[Web Server Overview](#)

[Hyper Text Transfer Protocol](#)

# WebServ Data Structures

## WS\_REQUEST

WS\_REQUEST is the typedef name for the \_WS\_REQUEST data structure. WS\_REQUEST holds all information pertaining to the HTTP request.

```
struct _WS_REQUEST {
    WS_TOKEN_INFO ws_pg_args;
    WS_TOKEN_INFO ws_ssi_args;
    WS_TOKEN_INFO ws_cookie_jar;
    WS_SERVER      *ws_server;
    WS_REQ_DATA    ws_rdata;
    CHAR           *ws_fname;
    UINT8          ws_ip[WS_IP_SIZE];
    UINT8          *ws_server_ip;
    INT            ws_sd;
    INT16          ws_method;
    INT            *ws_http_ver;
    CHAR           ws_user[WS_AUTH_NAME_LEN];
    CHAR           ws_ssi_buf[WS_SSI_LINE];
    UNSIGNED       *ws_ssl;
    INT16          ws_family;
};

typedef struct _WS_REQUEST WS_REQUEST;
```

The members of the structure are defined in [Table 12-1](#).

**Table 12-1. WS\_REQUEST**

Member	Description
ws_pg_args	Plugin arguments.
ws_ssi_args	SSI arguments.
ws_cookie_jar	Cookie information.
ws_server	Pointer to info for this server.
ws_rdata	Per-request allocated data.
ws_fname	Pointer to the URI (filename) of the requested entity.
ws_ip	IP address of the client.
ws_server_ip	Pointer to the IP address of the server.
ws_sd	Socket descriptor for this connection.
ws_method	HTTP method of this request GET, POST, HEAD etc.
ws_http_ver	Pointer to the version of HTTP that the client is using.
ws_user	Name of the authenticated user.

**Table 12-1. WS\_REQUEST (cont.)**

Member	Description
ws_ssi_buf	Buffer to hold an SSI Tag.
ws_ssl	Pointer to SSL structure, if enabled
ws_family	The family type to be used for the connection. Accepted values are: <ul style="list-style-type: none"><li>• NU_FAMILY_IP for an IPv4 connection</li><li>• NU_FAMILY_IP6 for an IPv6 connection</li></ul>

## Related Topics

[WebServ Data Structures](#)

[WebServ Functions](#)

# WebServ Functions

The following set of API functions allow you to utilize Nucleus WebServ. If there is ever a question as how to implement a specific function, it is always best to review the source code provided.

To use these APIs, include *os/include/networking/nu\_networking.h* in your application.

This section describes all of the functions that are used to interface with Nucleus WebServ.

- [CFS\\_Compress](#)
- [CFS-Decompress](#)
- [HTTP\\_Cookie\\_Value\\_By\\_Name](#)
- [HTTP\\_Make\\_Mime\\_Header](#)
- [HTTP\\_Redirect\\_Client](#)
- [HTTP\\_Send\\_Status\\_Message](#)
- [HTTP\\_Set\\_Cookie](#)
- [HTTP\\_Token\\_Value\\_by\\_Name](#)
- [HTTP\\_Token\\_Value\\_by\\_Number](#)
- [HTTP Uri\\_To\\_Url](#)
- [HTTPS\\_URI\\_To\\_URL](#)
- [WS\\_Find\\_Token\\_by\\_Case](#)
- [WS\\_In\\_String](#)

- [WS\\_Register\\_Plugin](#)
- [WS\\_Webserv\\_Initialize](#)
- [WS\\_Set\\_Auth\\_Callback](#)
- [WSC\\_Get\\_Use\\_Hostname](#)
- [WSC\\_Set\\_Use\\_Hostname](#)
- [WSF\\_File\\_Request\\_Status](#)
- [WSF\\_Read\\_File](#)
- [WSF\\_Write\\_File\\_System](#)
- [WSN\\_Read\\_Net](#)
- [WSN\\_Write\\_To\\_Net](#)

## CFS\_Compress

This function takes a file as input, compresses it and, then, stores it to outbuf. GIF and JPEG images are already compressed and will not compress further, and should not be sent to this function.

### Usage

```
INT CFS_Compress (INT  mode,
                  CHAR *inbuf,
                  CHAR *outbuf,
                  INT  length);
```

### Arguments

- mode

Determines if this function outputs a file.

WS\_DONT\_OUTPUT

Function does not compress a file. The function returns the size a file would be after compression. Useful to determine if a file needs to be compressed.

WS\_DO\_OUTPUT

Function does actual compression and places compressed file at outbuf.

- inbuf

Pointer to the data to be compressed.

- outbuf

Pointer to a buffer the compressed data will be placed.

- length

Length of the data file to be compressed.

### Return Values

- The size of the compressed file.

### Example

The following example is from UPL\_Compress, within *upl\_plgn.c*.

```
size = CFS_Compress(WS_DONT_OUTPUT, filemem, NU_NULL, (INT)length);
/* do not bother compressing unless there is some size benefit. */
if((i + WS_CHD_SZ ) < (length - 16))
{
    status = NU_Allocate_Memory(&System_Memory, (VOID*)&nbuf,
                               i + WS_CHD_SZ, NU_NO_SUSPEND);
    if(status != NU_SUCCESS)
    {
        return(WS_FAILURE);
    }
    /* Do the real compression. */
    if(size != CFS_Compress(WS_DO_OUTPUT, filemem, buf, length))
```



```
{
    #ifdef NU_WEBSERV_DEBUG
        printf("Compression Phase error\n");
    #endif
    return(WS_FAILURE);
}
```

## Related Topics

[WebServ Functions](#)

## CFS\_Decompress

This function takes a compressed file as input, decompresses it and, then, sends it across the network to the client.

### Usage

```
INT CFS_Decompress (WS_REQUEST *req,  
                   CHAR          *inbuf,  
                   CHAR          *outbuf,  
                   INT32         inlen);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- inbuf  
Pointer to the compressed file.
- outbuf  
Pointer to a buffer the decompressed data is placed. This is currently not being used, but is saved for future use.
- inlen  
Length of the input file.

### Return Values

- NU\_SUCCESS  
Upon successful completion.

### Example

The following example is taken from HTTP\_Serve\_File, within *http\_psr.c*.

```
/* Check if the compression identifier is at the beginning  
 * of the file. Be sure not to include the \0 in the compare.  
 */  
if(strncmp(req->ws_stat.ws_address, WS_CHD, WS_CHD_SZ) != 0)  
{  
    if(WSN_Write_To_Net(req, req->ws_stat.ws_address,  
        (UINT32)req->ws_stat.ws_size, WS_FILETRNSFR) != NU_SUCCESS)  
        NERRS_Log_Error (NERR_SEVERE, __FILE__, __LINE__);  
}  
else  
    CFS_Decompress(req, req->ws_stat.ws_address + 4, NU_NULL,  
        req->ws_stat.ws_size - 4);
```

## Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Cookie\_Value\_By\_Name

This function then returns a char pointer into the header where the value is located.

### Usage

```
CHAR HTTP_Cookie_Value_by_Name (CHAR      *name,  
                                WS_REQUEST *req);
```

### Arguments

- name  
Pointer to the NAME to which VALUE should be returned.
- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.

### Return Values

- A pointer to the VALUE string  
If NAME is found.
- A NULL pointer  
If NAME is not found.

### Description

It is possible to attach data to the HTTP packet header. This can be done by both the browser and the server. This data can be used to save the state of the client. Cookies are represented as name-value pairs, and more than one cookie can be placed within the header. To retrieve a cookie, the name must be known and passed into this function.

### Example

The following example is from JS\_Login, within *js\_auth.c*.

```
CHAR      *user;  
CHAR      *id;  
  
/* Get pointers to the cookies provided by the client. */  
user = HTTP_Token_Value_by_Name("user", req);  
id = HTTP_Cookie_Value_by_Name("id", req);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Make\_Mime\_Header

This function creates the HTTP packet header for a response.

### Usage

```
VOID HTTP_Make_Mime_Header (WS_REQUEST *req,  
                           INT32      length);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- length  
Total length of the file being sent.

### Description

This response is set to 200. By HTTP standards, this is the successful response code. This function should only be called when sending a file from memory. This creates the header used when `WSN_Write_To_Net` is called with the `FILETRANSFR` mode.

#### Caution



When using the more common mode usage of `WS_PLUGIN_DATA` and `WS_PLUGIN_SEND`, this should not be used.

### Example

```
/* Generate the response header for the transfer. */  
HTTP_Make_Mime_Header(req, req->ws_stat.ws_size);  
  
/* Send the file. */  
WSN_Write_To_Net(req, req->ws_stat.ws_address,  
                 (UINT32)req->ws_stat.ws_size, WS_FILETRNSFR)
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Redirect\_Client

This function sends a redirect command to the client.

### Usage

```
VOID HTTP_Redirect_Client (WS_REQUEST *req,  
                           CHAR        *url);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- url  
URL to which the client will be directed.

### Example

The following example illustrates the use of this API.

```
CHAR* url_buf  
  
/* Create the URL to which to redirect the client. */  
HTTP_Uri_To_Url(req, "/ssi.ssi", url_buf);  
/* Redirect the client to the new URL. */  
HTTP_Redirect_Client(req, url_buf);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Send\_Status\_Message

At times it is necessary to send the client a message. This message can state that the requested action was completed, that an error occurred, or the client does not have access to this plug-in.

### Usage

```
VOID HTTP_Send_Status_Message (WS_REQUEST *req,  
                               INT          code,  
                               CHAR         *mes);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- code  
HTTP status code. This should be a valid status from the HTTP standards document.
- mes  
Pointer to the message that is sent. This can be text or html.

### Description

Whatever the message may be, it can be sent through this function.

This function creates a header using the HTTP status code passed in appends the message to the header, and, then, sends the entire packet. The client receives the message and most likely displays the message. How this is displayed can vary between browsers.

### Example

This example is taken from HTTP\_Process\_Get within *http\_psr.c*.

```
/* The Message Body for HTTP 500 Internal Server Error */  
static const CHAR HTTP_Text_500a[]="\n  
<HEAD><TITLE>500 Internal Server Error</TITLE></HEAD>\r\n\  
<BODY><H1>500 Internal Server Error</H1>\r\n\  
Error: '";  
  
static const CHAR HTTP_Text_500b[]="' </BODY>";  
.  
.  
.  
  
strcpy(buf, HTTP_Text_500a);  
strcat(buf, "Plugin Failure");  
strcat(buf, HTTP_Text_500b);  
  
HTTP_Send_Status_Message(req, WS_PROTO_SERVER_ERROR, buf);
```

## Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)



## HTTP\_Set\_Cookie

Cookies are used to keep track of the state of the client, without using extra resources on the server.

### Usage

```
VOID HTTP_Set_Cookie (WS_REQUEST *req,  
                     CHAR        *name,  
                     CHAR        *value);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- name  
Pointer to the name, or identifier, of the cookie.
- value  
Pointer to the value to which this identifier is set.

### Description

The cookie is attached to the HTTP header and can be used by either the server plug-in, or by the client through Java script.

Cookies come in pairs, name=value. More than one cookie can be set within the header, but should be no more than `WS_COOKIE_HEAP` found in *os/include/networking/ws\_cfg.h*. As long as the client sends requests to the server, it sends all cookies that have been set, it is possible, however, to change the value of a cookie.

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Token\_Value\_by\_Name

When the client calls the plug-in, the client may send parameters to the plug-in function.

### Usage

```
CHAR* HTTP_Token_Value_by_Name (CHAR      *name,  
                                WS_REQUEST req);
```

### Arguments

- name  
Pointer to the NAME to which VALUE should be returned.
- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.

### Return Values

- A pointer to the VALUE string  
If NAME is found.
- A NULL pointer  
If NAME is not found.

### Description

These parameters are sent in the form NAME=VALUE&NAME=... To retrieve the VALUE, the NAME is passed to this function. If NAME exists within the string of parameters, a pointer to VALUE is returned, if not, a NULL pointer is returned.

#### Note



This API is actually HTTP\_Value\_by\_Name, which is not considered a user level API. HTTP\_Value\_by\_Name takes the two parameters described here, and a third, which describes which token array to search.

### Example

The following example illustrates the use of this API.

```
CHAR      *pg_string;  
pg_string = HTTP_Token_Value_by_Name("RADIOB", req);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP\_Token\_Value\_by\_Number

When the client calls the plug-in, the client may send parameters to the plug-in function.

### Usage

```
CHAR* HTTP_Token_Value_by_Number (INT      count,
                                   WS_REQUEST *req,
                                   UINT8     mode);
```

### Arguments

- **count**  
The position in the list of parameters that the desired value is requested.
- **req**  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- **mode**  
Specifies which token string to search. If the mode WS\_SSI is passed, the SSI token string is searched. If anything else is passed, the token array sent from the client is searched.

### Return Values

- A pointer to the VALUE string  
If count is found.
- A NULL pointer  
If count is out of range.

### Description

These parameters are sent in the form NAME=VALUE&NAME=... If it is known ahead of time the order in which the NAME/VALUE pairs will be passed, it is possible to speed up processing by asking for a particular position in the list. A number represents this position. If the number is valid, a pointer to VALUE is returned, if not, a NULL pointer is returned.

### Example

```
CHAR      *pg_string;
pg_string = HTTP_Token_Value_by_Number(3, req);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTP Uri To Uri

Converts a URI to an URL. The URI is a local path name where the URL contains the full path to the server along with the path to the file.

### Usage

```
VOID HTTP Uri To Uri (WS_REQUEST *req,  
                     CHAR *uri,  
                     CHAR *url_buf);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- uri  
Pointer to the URI file to be attached to the URL.
- url\_buf  
Pointer to the buffer to hold the converted URI.

### Example

The following example illustrates the use of this API.

```
CHAR* url_buf  
  
/* Create the URL to which to redirect the client.*/  
HTTP Uri To Uri(req, "/ssi.ssi", url_buf);  
/* Redirect the client to the new URL. */  
HTTP_Redirect_Client(req, url_buf);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## HTTPS\_URI\_To\_URL

This function is only used for SSL purposes. This function call is very similar to that of `HTTP_URI_To_URL` in which it creates a URL from a local path. However, it places an `HTTPS` within the URL stating that this is a link to a SSL secure location. When SSL is not present, the function call is precisely the same as `HTTP_URI_To_URL`.

### Usage

```
VOID HTTPS_Uri_To_Url (WS_REQUEST *req,  
                      CHAR        *uri,  
                      CHAR        *url_buf);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure `WS_REQUEST` is defined in the [WebServ Data Structures](#) section of this chapter.
- uri  
Pointer to the URI file to be attached to the URL.
- url\_buf  
Pointer to the buffer to hold the converted URI.

### Example

The following example illustrates the use of this API.

```
CHAR* url_buf  
  
/* Create the URL to which to redirect the client.*/  
HTTPS_Uri_To_Url(req, "/ssi.ssi", url_buf);  
/* Redirect the client to the new URL. */  
HTTP_Redirect_Client(req, url_buf);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## WS\_Find-Token\_by\_Case

This function returns a pointer to within the file where the occurrence token exists.

### Usage

```
CHAR* WS_Find-Token_by_Case (CHAR      *token,  
                             CHAR HUGE *file,  
                             CHAR      *last,  
                             UINT8     mode);
```

### Arguments

- token  
Pointer to the value being searched for.
- file  
Pointer to the data to be searched.
- last  
Pointer to the end of the data being searched.
- mode  
Flag to determine case sensitive search.

### Return Values

- A pointer to the position in the file where the token exists  
If the token is found.
- A NULL pointer  
Otherwise.

### Description

If token does not exist within the file, a NULL pointer is returned. This is similar to HTTP\_In\_String, however, this function is useful when the file is not NULL terminated. This can be used for both a case sensitive and case insensitive search.

### Example

```
CHAR line[elements];  
CHAR* s;  
  
s = WS_Find-Token("string", line, line[elements]);  
if(s == NU_NULL)  
    /* Token is not present in the string. */
```

### Related Topics

[WebServ Functions](#)

## WS\_In\_String

This function returns a pointer to within a string where the occurrence of the target string exists.

### Usage

```
CHAR *WS_In_String (CHAR      *target,  
                   CHAR HUGE *string );
```

### Arguments

- target  
Pointer to the string sequence to find.
- string  
Pointer to the string in which the search occurs.

### Return Values

- A pointer to the position in the string where the target exists  
If the target is found.
- A NULL pointer  
Otherwise

### Description

If target does not exist within string, a NULL pointer is returned. This is similar to HTTP\_Find\_Token, however, this function is useful when the data is NULL terminated.

### Example

```
CHAR line[elements]; /* NULL terminated string */  
CHAR* s;  
  
s = WS_In_String("string", line);  
if(s == NU_NULL)  
    /* "string" not present in string */
```

### Related Topics

[WebServ Functions](#)

## WS\_Register\_Plugin

This function registers a plug-in with the server.

### Usage

```
VOID WS_Register_Plugin (INT    (*plug_in) (WS_REQUEST *),  
                        CHAR    *uri,  
                        UINT8   flags);
```

### Arguments

- **plug\_in**  
A pointer to the plug-in function.
- **uri**  
Pointer to the URI string that is associated with the plug-in.
- **flags**  
Describes different characteristics of the plug-in. The following are valid flags:

**NU\_NULL**

There are no special characteristics for this plug-in.

**WS\_PRIVATE**

This plug-in should be considered private and only allowed access by authenticated users.

### Description

Every plug-in must be registered this way for the server to acknowledge it. This function must not be called before `WS_Webserv_Initialize`.

### Example

The following is an example of how to use this API:

```
/* Be sure that task_change exists as a function. */  
WS_Register_Plugin(task_change, "task_change", WS_PRIVATE);
```

### Related Topics

[WebServ Functions](#)



## WS\_Webserv\_Initialize

This function allows an easy interface to hook into Nucleus Webserver. This function initializes the web server tasks and the queue. It also calls the service initialization function.

### Usage

```
STATUS WS_Webserv_Initialize(VOID);
```

### Return Values

- NU\_SUCCESS  
If initialization has completed successfully.

### Description

All the tasks necessary for Nucleus WebServ to process requests are created in this function. The necessary structures and volatile file system, if used, are also set up here.

### Example

The following example illustrates the usage of *WS\_Webserv\_Initialize*.

```
STATUS          status;

/* Initialize the Web server. */
status = WS_Webserv_Initialize();
if(status != NU_SUCCESS)
{
    printf ("Failed to initialize Nucleus WebServ.\n");
}
```

### Related Topics

[WebServ Functions](#)

## WS\_Set\_Auth\_Callback

During the authentication process, it may be necessary for the application layer to evaluate the client for some reason. This function allows for the setting of a callback that is executed for each user upon login. Passed to this callback function is the authentication structure associated with the client.

### Usage

```
VOID WS_Set_Auth_Callback (VOID (*function) (WS_AUTH*));
```

### Arguments

- **function**  
Pointer to the function that is called upon login.

### Related Topics

[WebServ Functions](#)

## WSC\_Get\_Use\_Hostname

This gets the value of the configuration variable which is used to determine whether to use hostname addressing or IP literal addressing when building a URL.

### Usage

```
STATUS WSC_Get_Use_Hostname (UINT32 *setting_p);
```

### Arguments

- `setting_p`  
Pointer to the variable that is set to the value of the global configuration variable. The variable pointed to is set to `NU_TRUE` or `NU_FALSE`.

### Return Values

- `NU_SUCCESS`  
Successful completion of the service.
- `NU_INVALID_PARM`  
The input parameter is invalid.

### Related Topics

[WebServ Functions](#)

## WSC\_Set\_Use\_Hostname

This sets the configuration variable to determine whether to use hostname addressing instead of IP literal addressing when building a URL.

### Usage

```
STATUS WSC_Set_Use_Hostname (UINT32 setting);
```

### Arguments

- **setting**  
Turns the use of hostname on or off (NU\_TRUE or NU\_FALSE).

### Return Value

- **NU\_SUCCESS**  
Upon successful completion.

### Related Topics

[WebServ Functions](#)

## WSF\_File\_Request\_Status

This function searches for a particular URI.

### Usage

```
INT WSF_File_Request_Status (WS_REQUEST *req);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The file being searched for was found.
- WS\_FAILURE  
The file being searched for was not found.

### Description

It begins its search by looking at the req->ws\_fname string. The function first looks to see if the file is a plug-in. If it is, it sets the flag:

```
req->ws_stat->plugin = plugin function pointer;  
req->ws_stat->ws_flags = WS_PLUGIN;
```

If not, then it checks if the file is in memory. If the file is in memory, the following is set:

```
req->ws_stat->ws_address = pointer to start of file;  
req->ws_stat->ws_size = file length;  
req->ws_stat->ws_flags = WS_INCORE|WS_FOUND;
```

If not, the function proceeds to check if the file is on external storage. If it is, the following is set:

```
req->ws_stat->ws_size = length;  
req->ws_stat->ws_flags = WS_FOUND;
```

If this search passes, it is determined whether this file is to be treated as a secure file. If this is a private file, the following is set:

```
req->ws_stat.ws_flags |= WS_PRIVATE;
```

If these checks do not pass, then it returns a value that states that the URI is not found.

### Example

The following example is taken from UPL\_Save\_Upload\_File, within *upl\_plgn.c*.

```
CHAR * fname /* This is the file name requested. */
```

```
/* WSF_File_Request_Status expects to find file name here */
req->ws_fname = fname;

i = WSF_File_Request_Status(req);
```

## Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## WSF\_Read\_File

This function gets the file specified within the req structure from external storage. It then stores the file in the buffer.

### Usage

```
INT WSF_Read_File (WS_REQUEST *req,  
                  CHAR *buffer );
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- buffer  
A pointer to the buffer to hold the file read from external storage.

### Return Values

- NU\_SUCCESS  
The file was successfully copied into memory.
- WS\_FAILURE  
The file was not copied into memory.

### Example

This example is from HTTP\_Process\_SSI, within *http\_psr.c*.

```
/* Must get some memory into which the file is placed. */  
status = NU_Allocate_Memory(&System_Memory, (VOID*)&file,  
                           (UNSIGNED)req->ws_stat->ws_size, NU_NO_SUSPEND);  
  
if(status != NU_SUCCESS)  
{  
    return;  
}  
  
WSF_Read_File(req, file);
```

### Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)

## WSF\_Write\_File\_System

This function stores a file to external storage. The file can then be accessed through the use of fname.

### Usage

```
INT WSF_Write_File_System (CHAR    *fname,  
                           CHAR    *filemem,  
                           UINT32  length);
```

### Arguments

- fname  
Pointer to the name of the file.
- filemem  
Pointer to the file in memory.
- length  
Length of the file.

### Return Values

- NU\_SUCCESS  
File was successfully written.
- WS\_FAILURE  
The file was not written successfully.

### Related Topics

[WebServ Functions](#)



## WSN\_Read\_Net

Receives data from the network.

### Usage

```
INT WSN_Read_Net (WS_REQUEST *req,  
                  CHAR        *buf,  
                  UINT16      sz,  
                  UNSIGNED     timer);
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- buf  
A pointer to the buffer to hold the information read from the network connection.
- sz  
Size of the buffer.
- timer  
Length of time, in ticks, that Nucleus WebServ monitors for data.

### Return Values

- The number of bytes transferred.  
On successful completion.  
Otherwise, a Nucleus status code is returned. The status codes returned and their associated meanings are defined as follows. These values are described within the [NET](#) chapter.
  - NU\_NOT\_CONNECTED  
The connection is broken for some reason. Stop using the socket; it is best to close it.
  - NU\_INVALID\_SOCKET  
The socket parameter was not a valid socket value or it had not been previously allocated via the `NU_Socket` call.
  - NU\_NO\_PORT\_NUMBER  
No local port number was stored in the socket descriptor.

### Description

The amount of data received is returned as the number of bytes. If there was an error processing, an error status code is returned.

## Example

The following example illustrates the use of this API.

```
INT i;

i = WSN_Read_Net(req, req->ws_rdata->ws_lbuf, WSN_RECEIVE_SIZE);
if (i < 0)
{
    /* An Error has occurred. */
    return(WSN_REQ_NOACTION);
}
```

## Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WSN\\_REQUEST](#)

## WSN\_Write\_To\_Net

WSN\_Write\_To\_Net is called when data is to be sent over the network.

### Usage

```
STATUS WSN_Write_To_Net (WS_REQUEST *req,  
                        CHAR HUGE *buf,  
                        UINT32 sz,  
                        STATUS mode) ;
```

### Arguments

- req  
Pointer to all information pertaining to the HTTP request. Data structure [WS\\_REQUEST](#) is defined in the [WebServ Data Structures](#) section of this chapter.
- buf  
Pointer to the buffer holding the data to be sent to the network layer.
- sz  
The size of the buffer to be written out.
- mode  
The action for the function to take.

### Function Modes

The following summarizes the valid mode parameters.

- WS\_FILETRANSFER  
Regular File Transfer.
- WS\_REDIRECTION  
Redirection to another URL.
- WS\_PLUGIN\_DATA  
Plug-in or SSI Data. Adds data to the already created HTTP packet.
- WS\_PLUGIN\_SEND  
Send the PLUGIN or SSI data. Completes the packet and sends the packet across the network.

### Return Values

- NU\_SUCCESS  
Upon successful completion.  
Any other value means not enough memory for the buffer.

## Description

If the data is already in a valid HTTP packet form, FILETRANSFR can be used. If the buffer is full of raw data, or an incomplete packet PLUGINDATA, the PLUGINSEND combination should be used. When sending a file the packet header should be sent through HTTP\_Initiate\_Response.

## Example

The following example is from HTTP\_Parse\_Request, within *http\_psr.c*.

```
/* If a method comes in that is not supported send the 501 method
 * Not implemented message. */
strcpy(req->ws_response, HTTP_Text_H501);
strcpy(buf, HTTP_Text_D501);
len = strlen(buf);
HTTP_Initiate_Response(req, WS_PLUGIN_PROTO);
WS_Write_To_Net(req, buf, len, WS_PLUGIN_DATA);
WS_Write_To_Net(req, 0, 0, WS_PLUGIN_SEND);
```

## Related Topics

[WebServ Functions](#)

[WebServ Data Structures](#)

[WS\\_REQUEST](#)















































## WebSocket Implementation Overview

The ReadyStart implementation of the WebSocket protocol complies with RFC 6455 to provide a complete WebSocket client/server implementation.

The WebSocket Server component operates alongside the [HTTP Lite](#) Server module, which passes new incoming WebSocket connections to the WebSocket Server.

## Limitations of this WebSocket Implementation

This WebSocket implementation does not offer:

- Client/Server support for Sec-WebSocket-Extensions
- Client/Server Proxy support

## User Configuration Options

You can configure the following options in the Nucleus Configuration Editor (UI editor):

- `srv_stack_size`  
Size, in bytes, of the WebSocket Server task stack.
- `svr_stack_prio`  
The priority of the WebSocket Server stack task.
- `clnt_timeout`  
The number of seconds to wait for a response from the foreign server when issuing a WebSocket connection request as a client.
- `clnt_close_timeout`  
The number of seconds to wait for the foreign server to close the underlying TCP connection after a close frame has been sent and received by both sides before the client closes the underlying TCP connection.
- `ping_message`

The message to be sent in PING frames transmitted by the module periodically via `NU_WSOX_Schedule_Ping()`.

- `recv_handle_modify`

When a new WebSocket handle is created, receive notifications will be either enabled or disabled on that handle according to this value. This can be changed at run-time on a per handle basis via the API routine `NU_WSOX_Toggle_Recv()`.

For more information about creating a custom configuration, see “Creating a Custom Configuration”, in the chapter “Getting Started with Nucleus ReadyStart Using Sourcery Code Bench”, in the *Nucleus ReadyStart Guide*.

## Directory Structure

The source code for the client and server is stored at:  
`<install_path>\readystart\nucleus\os\networking\websocket`.

The header files are stored at: `<install_path>\readystart\nucleus\os\include\networking`.

### Related Topics

[WebSocket Implementation Overview](#)

[Limitations of this WebSocket Implementation](#)

## Creating a WebSocket Handle from the Application

To create a WebSocket handle, invoke `NU_WSOX_Create_Context`. This returns a 32-bit handle representing the context structure. This routine is used for both client and server connections. When creating a client handle, the connection will be initiated with the foreign server by this routine, and the routine will not return until the connection request is answered or the request times out.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[WebSocket/HTTP Dual Stack Implementation](#)

## WebSocket/HTTP Dual Stack Implementation

The WebSocket implementation uses the HTTP Lite Server to accept new WebSocket connections using the routines described in this section. An application can omit the HTTP Lite Server and create an implementation-specific server to service incoming connections using these same routines. If using the HTTP Lite Server to accept new WebSocket connections, the application must register the "websocket" upgrade plug-in routine `NU_WSOX_Process_Upgrade_Request` with the HTTP Lite Server using the routine `NU_HTTP_Lite_Register_Upgrade_Plugin`.

## HTTP Glue Layer

The [NU\\_WSOX\\_Accept](#) routine is used to accept incoming WebSocket connections over an existing HTTP Lite Server connection, but can also be used by the application to accept a connection from a protocol other than HTTP.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[Creating a WebSocket Handle from the Application](#)

## WebSocket Application Interface for Sending/Receiving Messages

This sections describes:

- [Sending Messages](#)
- [Receiving Messages](#)
- [Closing the Connection](#)
- [Sending PING Frames](#)
- [WebSocket Socket Options](#)

## Sending Messages

To send a message to the foreign side of the WebSocket connection, use [NU\\_WSOX\\_Send](#).

If the message is a fragment, note the following rules per RFC 6455:

- The fragments of one message must not be interleaved between the fragments of another message. The routine [NU\\_WSOX\\_Send](#) has no way to verify this, so you must ensure that your application follows this rule.
- Control frames cannot be fragmented.
- Since control frames cannot be fragmented, the type for all fragments in a message must be either text, binary, or one of the reserved opcodes.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[Receiving Messages](#)

## Receiving Messages

To receive a message, the client application registers a routine for incoming message notification when creating the WebSocket handle using the function [NU\\_WSOX\\_Create\\_Context](#). When the WebSocket stack receives data destined for an open handle, it invokes the `onmessage()` routine registered for that handle, notifying the application of the number of bytes in the received frame and the opcode of the data.

The application calls [NU\\_WSOX\\_Recv](#) from the context of the `onmessage()` routine to receive the data in an application-allocated buffer. [NU\\_WSOX\\_Recv](#) may be called only from the context of the `onmessage()` routine and returns an error if called at any other time. [NU\\_WSOX\\_Recv](#) tracks the number of bytes copied from the WebSocket and, if upon return from `onmessage()` there is data remaining on the WebSocket, `onmessage()` will be invoked again with the remaining data.

If the application cannot process all bytes available, it may invoke [NU\\_WSOX\\_Toggle\\_Recv](#) to disable successive calls to `onmessage()`. When the application is able to receive data on that handle, it should again invoke [NU\\_WSOX\\_Toggle\\_Recv](#), and the WebSocket stack will deliver the remaining data.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[Sending Messages](#)

## Closing the Connection

Either side may close the connection at any time, by using [NU\\_WSOX\\_Close](#).

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[Sending PING Frames](#)

## Sending PING Frames

PING frames can be configured to be transmitted periodically by the underlying software via the [NU\\_WSOX\\_Schedule\\_Ping](#) API routine.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[Closing the Connection](#)

## WebSocket Socket Options

The application may want to configure various socket options over the underlying TCP socket for the WebSocket handle. To accomplish this, the following two routines:

[NU\\_WSOX\\_Setsockopt](#) and [NU\\_WSOX\\_Getsockopt](#) have been created that map explicitly to the underlying [NU\\_Setsockopt](#) (IPv4/IPv6) and [NU\\_Getsockopt](#) (IPv4/IPv6) routines within the networking stack.

### Related Topics

[WebSocket Application Interface for Sending/Receiving Messages](#)

[WebSocket APIs](#)

## WebSocket Data Structures

This section describes the data structure used by the [WebSocket APIs](#).

- [NU\\_WSOX\\_CONTEXT\\_STRUCT](#)

### NU\_WSOX\_CONTEXT\_STRUCT

NU\_WSOX\_CONTEXT\_STRUCT is the typedef name for the \_NU\_WSOX\_CONTEXT\_STRUCT data structure which contains the connection parameters for the new handle.

```
typedef struct _NU_WSOX_CONTEXT_STRUCT
{
    VOID    (*onopen)(UINT32 handle, struct addr_struct *foreign_addr);
    VOID    (*onmessage)(UINT32 handle, WSOX_MSG_INFO *msg_info);
    VOID    (*onerror)(UINT32 handle, STATUS error, CHAR *reason);
    VOID    (*onclose)(UINT32 handle, STATUS error);
    CHAR    *host;
    CHAR    *resource;
    CHAR    *protocols;
    CHAR    *origins;
    CHAR    *ssl_ca;
    CHAR    *ssl_cert;
    CHAR    *ssl_key;
    CHAR    *extensions;
    UINT32  flag;
    INT     ssl_ca_size;
    INT     ssl_cert_size;
    INT     ssl_key_size;
    INT     port;
    UINT16  max_connections;
    UINT18  padN[2];
} NU_WSOX_CONTEXT_STRUCT;
```

The members of this structure are described in [Table 13-1](#). Some of the members were omitted as they are used internally.

**Table 13-1. NU\_WSOX\_CONTEXT\_STRUCTURE Members**

Member	Description
callback routines	<p>These routines must complete in a reasonable amount of time:</p> <pre>1  VOID onopen (UNIT32 handle,                struct addr_struct *foreign_addr);</pre> <p>This routine is invoked when the WebSocket transitions to the OPEN state.</p> <pre>2  VOID onmessage (UINT32 handle, WSOX_MSG_INFO                   *msg_info);</pre> <p>This routine is invoked when a message is received on the WebSocket. WSOX_MSG_INFO holds a data_len indicating the number of bytes in the received frame, an opcode field indicating the opcode of the incoming message, and a flags field that indicates whether the frame is part of a fragment.</p> <pre>3  VOID onerror (UINT32 handle, STATUS error,                 CHAR *reason);</pre> <p>This routine is invoked when an error has occurred on the WebSocket.</p> <pre>4  VOID onclose (UINT32 handle, STATUS error);</pre> <p>This routine is invoked when the WebSocket transitions to the CLOSED state.</p>
host	The part of the HTTP URI used to connect to this service as per section 3 of the RFC 6455. Since WebSocket should be loosely bound to the HTTP for extensibility with other protocols, this is not an HTTP URI.
resource	The resource name part of the HTTP URI used to access this service as per section 3 of RFC 6455.
protocols	Contains a comma separated list of application level protocols layered over the WebSocket protocol that are serviced by the same service routine, in order of preference.
origins	<p>If this is:</p> <ul style="list-style-type: none"> <li>• a client handle: The origin host name to be sent in connection requests</li> <li>• a server handle A comma separated list of origins to deny connection to the service or NU_NULL to allow any origin to connect.</li> </ul>
ssl_ca	A pointer to the certificate authority, if the client connection is made over SSL.

**Table 13-1. NU\_WSOX\_CONTEXT\_STRUCTURE Members (cont.)**

Member	Description
ssl_cert	A pointer to the certificate used if the server requests a certificate from the client.
ssl_key	A pointer to the key used if the server requests a certificate from the client.
extension	The optional protocol level extensions.
flag	<p>Flags for the WebSocket handle:</p> <ul style="list-style-type: none"> <li>• <b>NU_WSOX_SECURE</b> - Indicates a secure connection which will use “wss:” over HTTP.</li> <li>• <b>NU_WSOX_LISTENER</b> - This handle is a listener that can accept and service new incoming WebSocket connections.</li> <li>• <b>NU_WSOX_CERT_PEM_PTR</b> The certificate points to a PEM certificate in volatile memory.</li> <li>• <b>NU_WSOX_CERT_DER_PTR</b> The certificate points to a binary certificate in volatile memory.</li> <li>• <b>NU_WSOX_CERT_FILE</b> The certificate is stored on a local drive.</li> <li>• <b>NU_WSOX_KEY_PEM_PTR</b> The key sent to the foreign server points to a PEM key in volatile memory.</li> <li>• <b>NU_WSOX_KEY_DER_PTR</b> The key sent to the foreign server points to a binary key in volatile memory.</li> <li>• <b>NU_WSOX_KEY_FILE</b> The key sent to the foreign server is stored on a local drive.</li> <li>• <b>NU_WSOX_CA_PEM_PTR</b> The CA list points to a PEM certificate in volatile memory.</li> <li>• <b>NU_WSOX_CA_DER_PTR</b> The CA list points to a binary certificate in volatile memory.</li> <li>• <b>NU_WSOX_CA_FILE</b> The CA list is stored on a local drive.</li> <li>• <b>NU_WSOX_VERIFY_PEER</b> Verify the foreign server certificate when connecting via a client connection.</li> <li>• <b>NU_WSOX_NO_DOMAIN_CHECK</b> Do not check the domain name before connecting via SSL.</li> </ul>
ssl_ca_size	The size of the certificate authority, if ssl_ca is not pointing to a file. If it is pointing to a file, this parameter is ignored.
ssl_cert_size	The size of the certificate, if ssl_cert is not pointing to a file. If it is pointing to a file, this parameter is ignored.
ssl_key_size	The size of the key, if ssl_key is not pointing to a file. If it is pointing to a file, this parameter is ignored.

**Table 13-1. NU\_WSOX\_CONTEXT\_STRUCTURE Members (cont.)**

Member	Description
port	The port to connect to if the local node initiates the connection. Otherwise, all connections are serviced by port 80 or 443 (for TLS).
max_connections	The maximum number of clients that can be simultaneously connected to this listener handle.

---

**Note**



The members `ssl_ca`, `ssl_cert`, and `ssl_key` (and their corresponding size members), apply only to the routine [NU\\_WSOX\\_Create\\_Context](#) for a new client handle when the flag parameter is `NU_WSOX_SECURE`.

---

## WebSocket APIs

This section describes the WebSocket APIs used in the implementation:

- [NU\\_WSOX\\_Create\\_Context](#)
- [NU\\_WSOX\\_Accept](#)
- [NU\\_WSOX\\_Send](#)
- [NU\\_WSOX\\_Recv](#)
- [NU\\_WSOX\\_Toggle\\_Recv](#)
- [NU\\_WSOX\\_Close](#)
- [NU\\_WSOX\\_Schedule\\_Ping](#)
- [NU\\_WSOX\\_Setsockopt](#)
- [NU\\_WSOX\\_Getsockopt](#)
- [NU\\_WSOX\\_Process\\_Upgrade\\_Request](#)



## NU\_WSOX\_Create\_Context

Creates a new internal WebSocket context structure and returns the handle to the caller.

### Usage

```
STATUS NU_WSOX_Create_Context (NU_WSOX_CONTEXT_STRUCT *context_ptr,  
                               UINT32 *user_handle);
```

### Arguments

- context\_ptr  
A pointer to the structure on which to base the new structure.
- user\_handle  
A pointer that is filled in with the handle for the new [NU\\_WSOX\\_CONTEXT\\_STRUCT](#). This handle is used by the application to send/receive data.

### Return Values

- NU\_SUCCESS  
The function completed successfully and the new structure was created. If the handle is not a listener, the connection was successfully made with the foreign server.
- NU\_INVALID\_PARM  
An input parameter is invalid.
- Client specific return values:
  - WSOX\_INVALID\_HOST  
The host field is invalid.
  - WSOX\_TIMEOUT  
The server did not respond to the HTTP GET request.
  - WSOX\_CNXN\_ERROR  
The server rejected the connection.
  - WSOX\_PKT\_ERROR  
The server's response was invalid.
  - WSOX\_PROTOCOL\_ERROR  
The server does not support the requested protocol(s).
  - WSOX\_NO\_HANDLES  
There is not a free handle index in the system for the new structure.
  - WSOX\_SSL\_ERROR  
The SSL connection could not be established.

- Operating-system specific error

## Examples

### Server Handle:

```
STATUS                status;
NU_WSOX_CONTEXT_STRUCT wsox_lstnr;
UINT32                lstnr_handle;

/* Clear the NU_WSOX_CLIENT_STRUCT structure. */
memset(&wsox_lstnr, 0, sizeof(NU_WSOX_CONTEXT_STRUCT));

/* Set up a valid listener structure. */
wsox_lstnr.resource = "index.htm";
wsox_lstnr.protocols = "chat, chatty";
wsox_lstnr.flag = NU_WSOX_LISTENER;
wsox_lstnr.onclose = wsox_onclose;
wsox_lstnr.onerror = wsox_onerror;
wsox_lstnr.onmessage = wsox_onmessage;
wsox_lstnr.onopen = wsox_onopen;
wsox_lstnr.max_connections = 10;

/* Create the listener structure. */
status = NU_WSOX_Create_Context(&wsox_lstnr, &lstnr_handle);
```

### Client Handle:

```
STATUS                status;
NU_WSOX_CONTEXT_STRUCT wsox_cli;
UINT32                cli_handle;

/* Clear the NU_WSOX_CLIENT_STRUCT structure. */
memset(&wsox_lstnr, 0, sizeof(NU_WSOX_CONTEXT_STRUCT));

/* Set up a valid client structure. */
wsox_cli.resource = "/index.htm";
wsox_cli.protocols = "chat, chatty";
wsox_cli.onclose = wsox_onclose;
wsox_cli.onerror = wsox_onerror;
wsox_cli.onmessage = wsox_onmessage;
wsox_cli.onopen = wsox_onopen;
wsox_cli.host = "mentor.com";
wsox_cli.port = 80;

/* Create the client structure. */
status = NU_WSOX_Create_Context(&wsox_cli, &cli_handle);
```

## Related Topics

[WebSocket APIs](#)

## NU\_WSOX\_Accept

Performs server-side processing of an incoming connection request from a foreign client, and creates a new handle for the connection upon successful completion of the connection.

### Usage

```
STATUS NU_WSOX_Accept (NU_WSOX_CONTEXT_STRUCT *client_req,
                      INT socketd,
                      CHAR *key,
                      VOID *ssl);
```

### Arguments

- **client\_req**  
The structure on which to base the new structure that holds the data for the pending client request. The parameter `client_req->protocols` is modified to return only the protocols that are supported by the listening socket. The structure [NU\\_WSOX\\_CONTEXT\\_STRUCT](#) is described in the [WebSocket Data Structures](#) section.
- **socketd**  
The socket on which the new connection is being requested.
- **key**  
A pointer to the null-terminated Sec-WebSocket-Key used by the client in the connection request.
- **ssl**  
A pointer to the SSL connection structure if the connection is secure.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the connection was created.
- An operating-system specific error

### Description

This routine is currently called internally by the WebSocket/HTTP glue layer, but could be used at the application layer to accept connections received over protocols other than HTTP.

### Examples

```
NU_WSOX_CONTEXT_STRUCT context_ptr;
CHAR keys[16];
STATUS status;
INT socketd;
```

```
/* Parse the resource, host, origin, Sec-WebSocket-Protocol
 * and Sec-WebSocket-Key from the incoming request
 * header to populate the resource, host, origins, and protocols
 * fields of the data structure.
 */
...

status = NU_WSOX_Accept(&context_ptr, socketd, keys, NU_NULL);
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Create\\_Context](#)

## NU\_WSOX\_Send

This function sends data to the foreign side of the WebSocket connection.

### Usage

```
STATUS NU_WSOX_Send(UINT32  handle,
                    CHAR    *buffer,
                    UINT64  *data_len,
                    INT     opcode,
                    UINT8   flags);
```

### Arguments

- **handle**  
The WebSocket handle over which to send data.
- **buffer**  
A pointer to the buffer containing data to be sent. This buffer can be NU\_NULL if data length is zero.
- **data\_len**  
On input, it points to the length of the data in the buffer. On return, it points to the number of bytes successfully transmitted. This value can be zero if the buffer is NU\_NULL.
- **opcode**  
The type of data in the buffer. Valid values are:
  - WSOX\_TEXT\_FRAME - Text data
  - WSOX\_BINARY\_FRAME - Binary data
  - WSOX\_PING\_FRAME - Ping frame
  - WSOX\_PONG\_FRAME - Pong frame
- **flags**  
Flags associated with the transmission:
  - NU\_WSOX\_FRAGMENT  
There are more bytes to be included in the message than being transmitted with this function call.

### Return Values

- **NU\_SUCCESS**  
At least some of the data was successfully transmitted. Check the data\_len parameter to determine how much data was sent.
- **NU\_INVALID\_PARM**  
An input parameter is invalid.

- **WSOX\_INVALID\_HANDLE**  
The handle does not represent an open connection.
- **WSOX\_INVALID\_OPCODE**  
The caller is attempting to fragment a control frame.
- **WSOX\_CNXN\_ERROR**  
The underlying TCP connection has been closed.
- Operating-system specific error

## Examples

```
STATUS          status;  
UINT32          cli_handle;  
CHAR            buffer[16];  
UINT64          data_len = 16;  
  
status = NU_WSOX_Send(cli_handle, buffer, &data_len, WSOX_BINARY_FRAME,  
                      0);
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Recv](#)

## NU\_WSOX\_Recv

This routine receives data over a WebSocket handle.

### Usage

```
STATUS NU_WSOX_Recv(UINT32  handle,
                    CHAR    *buffer,
                    UINT64  *data_len);
```

### Arguments

- **handle**  
The handle over which to receive data as provided by the onmessage() routine.
- **buffer**  
The application allocated buffer into which to receive the data.
- **data\_len**  
Pointer to the size of the buffer passed into the routine on input. On return, pointer to the number of bytes copied into the buffer by the routine.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and at least some of the data was received.
- **NU\_INVALID\_PARM**  
An input parameter is NULL.
- **WSOX\_INVALID\_HANDLE**  
The handle does not represent an open connection.
- **WSOX\_INVALID\_ACCESS**  
The routine is being called outside the context of the registered onmessage() routine.
- **WSOX\_DECODE\_ERROR**  
The data is not properly encoded UTF-8 data. The connection will be closed.

### Examples

```
VOID wsox_onmessage(UINT32 handle, WSOX_MSG_INFO *msg_info)
{
    CHAR    *buffer;
    STATUS  status;

    /* Allocate a buffer for the data. */
    status = NU_Allocate_Memory(MEM_Cached, (VOID**)&buffer,
                               msg_info->data_len, NU_NO_SUSPEND);

    if (status == NU_SUCCESS)
    {
        status = NU_WSOX_Recv(handle, buffer, &msg_info->data_len);
    }
}
```

```
        /* Do something with the data. */  
        ...  
        NU_Deallocate_Memory(buffer);  
    }  
}
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Send](#)



## NU\_WSOX\_Toggle\_Recv

This function toggles receive notifications on the specified handle.

### Usage

```
STATUS NU_WSOX_Toggle_Recv (UINT32  handle,  
                           BOOLEAN enable);
```

### Arguments

- **handle**  
The handle for which to toggle receive notifications.
- **enable**  
NU\_TRUE  
To enable receive notifications on the handle.  
NU\_FALSE  
To disable receive notifications on the handle.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully.
- **WSOX\_INVALID\_HANDLE**  
The handle is invalid.
- Operating-system specific error code

### Description

When set to NU\_FALSE, the WebSocket master task will not invoke onmessage() when an incoming message is received on this connection. When set to NU\_TRUE, the WebSocket master task will resume invoking onmessage() when an incoming message is received.

Note that if the application does not remove all data from a handle when onmessage() is invoked, and receive notifications are not disabled for that handle from onmessage(), onmessage() will be immediately invoked again.

This routine may be invoked from the onmessage() callback routine registered with the respective handle or from any other application task.

### Examples

```
VOID wsox_onmessage(UINT32 handle, WSOX_MSG_INFO *msg_info)
{
    STATUS status;

    /* Disable receive notifications. */
    status = NU_WSOX_Toggle_Recv(handle, NU_FALSE);
}
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Send](#)

## NU\_WSOX\_Close

This function closes the local side of the connection, indicating that it has finished sending all data.

### Usage

```
STATUS NU_WSOX_Close(UINT32 handle,
                     UINT16 status_code,
                     CHAR *reason);
```

### Arguments

- **handle**  
The WebSocket handle representing the connection to close.
- **status\_code**  
The status code to include in the closed frame.
- **reason**  
The UTF-8 encoded string to include as the reason for the closure.

Valid reasons per RFC 6455 are:

```
#define WSOX_NORMAL 1000 /* Normal closure. */
#define WSOX_GOING_DOWN 1001 /* The endpoint is going down. */
#define WSOX_PROTO_ERROR 1002 /* Protocol error. */
#define WSOX_UNRECOGNIZED_DATA 1003 /* Does not recognize data type. */
#define WSOX_BAD_DATA 1007 /* The received data is not
    * consistent with the type of data.
    */
#define WSOX_POLICY_VIOLATION 1008 /* A message has violated the local
    * policy. */
#define WSOX_MESSAGE_TOO_BIG 1009 /* A message is too big to process.
    */
#define WSOX_MISSING_EXTENSIONS 1010 /* An extension was expected. */
#define WSOX_UNEXPECTED_COND 1011 /* An unexpected condition was
    * encountered. */
```

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the closed frame was sent.
- **NU\_INVALID\_HANDLE**  
The handle is invalid.
- **WSOX\_MSG\_TOO\_BIG**  
The reason is too big. The maximum length of a closed frame is 125 bytes.

- Operating-system specific error code

### Description

If a closed frame has not already been received, the application may continue to receive data over this handle. Otherwise, the connection is closed for both sending and receiving data.

### Examples

```
STATUS status;  
UINT32 cli_handle;  
  
status = NU_WSOX_Close(cli_handle, WSOX_NORMAL, NU_NULL);
```

### Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Accept](#)

## NU\_WSOX\_Schedule\_Ping

This routine schedules a PING control frame to be transmitted to the destination at the indicated rate. Any PONG control frame received triggers the onmessage() callback routine associated with the handle.

### Usage

```
STATUS NU_WSOX_Schedule_Ping(UINT32 handle,
                             UINT32 interval,
                             UINT32 delay,
                             UINT8  retrans_count);
```

### Arguments

- **handle**  
The handle for which to schedule the PING control frame transmission.
- **interval**  
The interval, in clock ticks, at which to transmit PING control frames. Setting this value to zero cancels a previous call to this routine.
- **delay**  
The delay, in clock ticks, between unanswered PING control frames. This value can be set to zero only if retrans\_count is also zero, which indicates that the PING frame will not be retransmitted when no PONG is received in the interval timeout.
- **retrans\_count**  
The number of unanswered PING frames to retransmit before notifying the application via the onerror() routine. If this value is set to zero, the delay input parameter is ignored.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the PING was scheduled.
- **WSOX\_INVALID\_HANDLE**  
The handle is invalid.
- **WSOX\_DUP\_REQUEST**  
A PING has already been scheduled for this handle. The caller must cancel an existing outstanding scheduled PING to change the interval parameters.
- **NU\_INVALID\_PARM**  
Delay is set to zero when retrans\_count is non-zero.

- Operating-system specific error

## Description

To disable PINGs on the connection, set interval to zero. Note that PINGs are disabled by default and must be enabled by the application upon completion of a successful connection over the handle.

## Examples

```
STATUS status;
UINT32 cli_handle;

/* Send a PING from the local client to the server every two minutes. */
status = NU_WSOX_Schedule_Ping(cli_handle, 2 * 60 * TICKS_PER_SECOND,
                               1 * TICKS_PER_SECOND, 5);
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Create\\_Context](#)

## NU\_WSOX\_Setsockopt

This routine invokes [NU\\_Setsockopt \(IPv4/IPv6\)](#) for the socket associated with the respective handle.

### Usage

```
STATUS NU_WSOX_Setsockopt (UINT32 handle,
                           INT     level,
                           INT     optname,
                           VOID    *optval,
                           INT     optlen);
```

### Arguments

- **handle**  
The handle for which to set the socket option.
- **level**  
The protocol level. For valid level options, refer to [NU\\_Setsockopt \(IPv4/IPv6\)](#).
- **optname**  
The options being set. For valid optname options, refer to [NU\\_Setsockopt \(IPv4/IPv6\)](#).
- **optval**  
Pointer to the new value for the option.
- **optlen**  
The size in bytes of the location pointed to by optval.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the option was set.
- **NU\_INVALID\_PARM**  
An input parameter is NULL.
- **WSOX\_INVALID\_HANDLE**  
The handle is invalid.
- Operating-system specific error

### Examples

```
STATUS status;
INT32  probe_to = 2 * 60 * TICKS_PER_SECOND;
UINT32 cli_handle;

/* Set the timeout value for transmitting Window Probes to 2 minutes. */
status = NU_WSOX_Setsockopt(cli_handle, IPPROTO_TCP, TCP_PROBE_TIMEOUT,
                           &probe_to, sizeof(INT32));
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSOX\\_Getsockopt](#)



## NU\_WSOX\_Getsockopt

This function invokes [NU\\_Getsockopt \(IPv4/IPv6\)](#) for the socket associated with the respective handle.

### Usage

```
STATUS NU_WSOX_Getsockopt (UINT32 handle,
                           INT     level,
                           INT     optname,
                           VOID    *optval,
                           INT     *optlen);
```

### Arguments

- **handle**  
The handle for which to get the socket option.
- **level**  
The protocol level. For valid level options, refer to [NU\\_Getsockopt \(IPv4/IPv6\)](#).
- **optname**  
The option to get. For valid optname options, refer to [NU\\_Getsockopt \(IPv4/IPv6\)](#).
- **optval**  
Pointer to the location where the option status can be read from.
- **optlen**  
The size in bytes of the location pointed to by optval.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the option was retrieved.
- **NU\_INVALID\_PARM**  
An input parameter is NULL.
- **WSOX\_INVALID\_HANDLE**  
The handle is invalid.
- **Operating-system specific error**

### Examples

```
STATUS status;
INT32  probe_to;
INT    probe_to_len = sizeof(INT32);
UINT32 cli_handle;

/* Get the timeout value for transmitting Window Probes. */
status = NU_WSOX_Getsockopt(cli_handle, IPPROTO_TCP, TCP_PROBE_TIMEOUT,
                           &probe_to, &probe_to_len);
```

## Related Topics

[WebSocket APIs](#)

[NU\\_WSIX\\_Setsockopt](#)

## NU\_WSOX\_Process\_Upgrade\_Request

This function parses the components of an HTTP upgrade request into the appropriate data structure format. The WebSocket module uses this data structure to complete and accept the incoming connection.

### Usage

```
STATUS NU_WSOX_Process_Upgrade_Request (INT  method,
                                         CHAR  *uri,
                                         CHAR  *buf_ptr,
                                         INT   buf_len,
                                         INT   socketd,
                                         VOID  *ssl);
```

### Arguments

- **method**  
The method(s) being used by the request:  
HTTP\_LITE\_PUT  
HTTP\_LITE\_POST  
HTTP\_LITE\_DELETE  
HTTP\_LITE\_GET
- **uri**  
The URI associated with the operation.
- **buf\_ptr**  
A pointer to the HTTP header buffer.
- **buf\_len**  
The length of the HTTP header buffer.
- **socketd**  
The socket for the connection.
- **ssl**  
The SSL structure associated with the new connection, if the connection is secure.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and the upgrade request was processed.
- **WSOX error code**  
The connection was not accepted. The caller does not need to do anything in response to an error code.

## Description

This routine is currently called internally by the WebSocket/HTTP glue layer, but could be used at the application layer to accept connections received over protocols other than HTTP.

## Examples

```
STATIC VOID HTTP_Lite_Process_Client (VOID)
{
    . . .

    /* If an upgrade request was received */
    if (HTTP_Session->flags & HTTP_UPGRADE_REQ)
    {
        /* Check if the upgrade type is associated with a valid
         * plug-in.
         */
        HTTP_Session->plug_ptr =
            HTTP_Lite_Get_Upgrade_Plugin(HTTP_Session->upgrade_req,
                                         http_method);

        if (HTTP_Session->plug_ptr)
        {
            /* Invoke the plug-in. */
            status = HTTP_Session->plug_ptr->uplugin(http_method, uri,
                                                    HTTP_Session->buffer,
                                                    HTTP_Session->in_hdr_size,
                                                    HTTP_Session->socketd,
                                                    HTTP_LITE_SVR_SSL_STRUCT);

            . . .
        }
    }
}
```

## Related Topics

[WebSocket APIs](#)





# Chapter 14

## Simple Network Time Protocol (SNTP)

---

### **Warning**



In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## SNTP Overview

SNTP is a simpler adaptation of Network Time Protocol (NTP), which is used to synchronize computer clocks over the Internet. NTP Version 4 is completely defined in RFC-5905. SNTP uses a client/server model and operates at the transport layer as an application protocol. The Nucleus SNTP client functions correctly with both SNTP Version 3 and 4 servers. Support for both the IPv4 and IPv6 addressing family is included.

Nucleus SNTP client is an embedded implementation of the SNTP Version 4 client, which is tuned for applications where memory and CPU resources are limited. Applications can use the SNTP Client to set a real-time clock, display the current time on some type of display device, save to a file system, and so on. A set of user-level functions are supplied to allow the application to set the time zone, Daylight Savings Time (DST), the SNTP server address, as well as retrieve the current time.

The Nucleus SNTP client is capable of retrieving the time from many NTP servers simultaneously, and using an algorithm to determine the server with the highest degree of accuracy. Additionally, the frequency of server updates can be configured. This implementation is a unicast client only; no support for broadcast/multicast servers is available in this version.

## NTP/SNTP Servers

The SNTP client can be used with both NTP and SNTP servers. NTP servers keep better time and are the main time servers used throughout the world. When configuring the Nucleus SNTP client, you should consider the distance from the server and polling frequency. A shorter distance from the server results in a more accurate time returned. The accuracy is usually in the millisecond range and, depending on your requirements, distance may not be an issue.

The polling frequency should be no more often than once every sixteen seconds. This, though, is dependant on the NTP/SNTP server and on your application requirements. Polling once or twice a day is enough to maintain the correct time.

See [www.ntp.org](http://www.ntp.org) for more information, domain names, IP addresses, and access rules for NTP servers.

## SNTP Interaction with Nucleus MMU

Nucleus SNTP Client has been instrumented with the appropriate 'hooks' to support the Nucleus MMU product. Nucleus MMU protects system resources such as code and data space from errant user level tasks as well as protecting individual user modules from each other. For more information about Nucleus MMU, contact your sales associate. There are no known incompatibility issues.

## SNTP Client Data Structures

This section describes the Nucleus SNTP-specific data structures used by the various API level routines. A description of each data structure and its corresponding members is provided.

### Note



The structures have read/write access unless otherwise stated.

- [SNTPC\\_TIME](#)
- [SNTPC\\_SERVER](#)

## SNTPC\_TIME

SNTPC\_TIME is the typedef for the structure `_sntp_time`.

This structure is used to keep time. It can be used to represent either local time or Prime Epoch time (0h, 1 January, 1900), depending on the context in which it is used.

```
typedef struct _sntp_time
{
    UINT32 sntp_seconds;
    UINT32 sntp_useconds;
    INT     sntp_is_dst;
} SNTPC_TIME;
```

The members of the structure are defined as in [Table 14-1](#).

**Table 14-1. SNTPC\_TIME**

Member	Description
sntp_seconds	The number of seconds since 12:00:00 AM January 1, 1900.
sntp_useconds	This is the number of microseconds, in addition to the seconds value (not "instead of"). It is used to provide finer time granularity.



**Table 14-1. SNTPC\_TIME (cont.)**

Member	Description
sntp_is_dst	NU_TRUE means that time has been adjusted for Daylight Savings Time.

## Related Topics

[SNTP Client Data Structures](#)[SNTPC\\_Get\\_Time](#)[SNTPC\\_Get\\_Time\\_From\\_Server](#)

## SNTPC\_SERVER

SNTPC\_SERVER is the typedef for the structure `_sntp_server`.

This structure is used to pass information about the NTP servers to and from the SNTPC API. Each server must have a unique address.

```
typedef struct _sntp_server
{
    struct addr_struct sntp_server_addr;
    CHAR               *sntp_server_hostname;
    UINT32              sntp_poll_interval;
    UINT16              sntp_src_port;
} SNTPC_SERVER;
```

The members of the structure are defined as in [Table 14-2](#).

**Table 14-2. SNTPC\_SERVER**

Member	Description
sntp_server_addr	The server's address. Each server added to the client must have a unique address.
sntp_server_hostname	This is an optional hostname of the server which will be resolved by DNS. If an IP address is specified in the "sntp_server_addr" member, then this hostname member should be set to NU_NULL. Otherwise, it should contain a null-terminated name of the server. Also note that the IP-family and port number must be specified in "sntp_server_addr" even if a hostname is given instead of IP address.
sntp_poll_interval	The minimum polling interval in seconds. A value of zero indicates the server should not be polled.
sntp_src_port	The UDP local port. Zero indicates "don't care".

## Related Topics

[SNTP Client Data Structures](#)

[SNTPC\\_Add\\_Server](#)

[SNTPC\\_Delete\\_Server](#)

[SNTPC\\_Server\\_Query](#)

# Nucleus SNTP Client - API Interface

This chapter discusses the API functions for Nucleus SNTP Client. Refer to `\include\networking\nu_networking.h` for accessing APIs and data structures of SNTP client component.

- [SNTPC\\_Add\\_Server](#)
- [SNTPC\\_Delete\\_Server](#)
- [SNTPC\\_Purge\\_Server\\_List](#)
- [SNTPC\\_Server\\_Query](#)
- [SNTPC\\_Set\\_Timezone](#)
- [SNTPC\\_Get\\_Timezone](#)
- [SNTPC\\_Get\\_Time](#)
- [SNTPC\\_Get\\_Time\\_From\\_Server](#)

## SNTPC\_Add\_Server

This function adds a new NTP server to the list of possible hosts. The added servers must each have a unique address, port, and family.

---

**Note**

The [SNTPC\\_SERVER](#) structure must be populated prior to calling the `SNTPC_Add_Server` function.

---

### Usage

```
STATUS SNTPC_Add_Server (SNTPC_SERVER *server);
```

### Arguments

- `server`

A pointer to a server structure that has been filled out.

[SNTPC\\_SERVER](#) is a structure defined in the [SNTP Client Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
Function completed successfully; server added.
- `SNTPC_ALREADY_EXISTS`  
Server already exists in the database.

### Description

The client must have at least one server to query; however, multiple servers can be added using this API. In the case of multiple servers, each server is independently queried and the time, relative to each server, is maintained and can be queried using the [SNTPC\\_Get\\_Time\\_From\\_Server](#) API function.

### Example

```
STATUS          status;
SNTPC_SERVER    server;
UINT8 serv_ip_addr[] = {192,168,1,1};

/* Set up the server structure. */
memcpy(&server.sntp_server_addr.id.is_ip_addrs, serv_ip_addr,
      sizeof(serv_ip_addr));
server.sntp_server_hostname = NU_NULL;
server.sntp_server_addr.family = NU_FAMILY_IP;
server.sntp_server_addr.port = 123;
server.sntp_poll_interval = 900;
server.sntp_src_port = 0;

status = SNTPC_Add_Server(&server);
```

Second example:

```
STATUS status;
SNTPC_SERVER server;

/* Set up the server structure. */
server.sntpc_server_hostname = "2.pool.ntp.org";
server.sntpc_server_addr.family = NU_FAMILY_IP;
server.sntpc_server_addr.port = 123;
server.sntpc_poll_interval = 1800;
server.sntpc_src_port = 0;

status = SNTPC_Add_Server(&server);
```

## Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Delete\\_Server](#)

## SNTPC\_Delete\_Server

This function removes a specific NTP server from the list of possible hosts. Once removed, the server is no longer be queried.

### Usage

```
STATUS SNTPC_Delete_Server (SNTPC_SERVER *server);
```

### Arguments

- server

A pointer to the server structure describing the server to be removed.

[SNTPC\\_SERVER](#) is a structure defined in the [SNTP Client Data Structures](#) section of this chapter.

### Return Values

- NU\_SUCCESS  
The function completed successfully; the server was removed.
- NU\_NO\_DATA  
The server does not exist in the database.

### Example

```
STATUS      status;  
SNTPC_SERVER server;  
  
/* Fill in "server" with server details */  
...  
status = SNTPC_Delete_Server(&server);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Add\\_Server](#)

[SNTPC\\_Purge\\_Server\\_List](#)

## SNTPC\_Purge\_Server\_List

This function quickly removes all the NTP servers in the database. The NTP server database is empty after this API is called.

### Usage

```
STATUS SNTPC_Purge_Server_List (VOID);
```

### Arguments

- None

### Return Values

- None

### Example

```
SNTPC_Purge_Server_List();
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Delete\\_Server](#)

## SNTPC\_Server\_Query

This function searches for the previously added server by address and fills out the passed in server structure with the server information.

### Usage

```
STATUS SNTPC_Server_Query(struct addr_struct *addr,  
                          CHAR               *server_hostname,  
                          SNTPC_SERVER      *ret_server);
```

### Arguments

- **addr**  
The address of the server.
- **server\_hostname**  
An optional parameter which should be NU\_NULL if an IP address is specified in the "addr" parameter, and should contain a hostname for the server if an IP has not been specified in "addr".
- **ret\_server**  
Pointer to a server structure to be filled out.  
[SNTPC\\_SERVER](#) is a structure defined in the [SNTP Client Data Structures](#) section of this chapter.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; server found and returned in ret\_server.
- **OS specific error**

### Example

```
STATUS          status;  
struct addr_struct addr;  
SNTPC_SERVER    server;  
UINT8 serv_ip_addr[] = {192,168,1,1};  
addr.family = NU_FAMILY_IP;  
addr.port = 123;  
  
/* Set up the server structure. */  
memcpy(&addr.id.is_ip_addrs, serv_ip_addr, sizeof(serv_ip_addr));  
  
status = SNTPC_Server_Query(&addr, NU_NULL, &server);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Add\\_Server](#)

## SNTPC\_Set\_Timezone

This function adjusts the timezone and DST values. Use the defined timezone values in *nucleus\os\include\networking\sntp.h* for the timezone parameter.

### Usage

```
STATUS SNTPC_Set_Timezone (INT16 timezone,  
                           UINT8  dst_enabled);
```

### Arguments

- **timezone**  
Minutes offset from UTC.
- **dst\_enabled**  
Boolean to adjust for DST if NU\_TRUE.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; timezone was set.
- **OS-specific error**

### Example

```
STATUS status;  
status = SNTPC_Set_Timezone(SNTPC_TIMEZONE_CDT, NU_TRUE);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Get\\_Timezone](#)



## SNTPC\_Get\_Timezone

This function returns the current settings for timezone and DST.

### Usage

```
STATUS SNTPC_Get_Timezone (INT16 *timezone,  
                           UINT8 *dst_enabled);
```

### Arguments

- **timezone**  
Pointer to a variable where the timezone value is returned (in minutes).
- **dst\_enabled**  
Pointer to a variable where the DST value is returned.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully; the timezone value and the DST were returned.
- **OS specific error**

### Example

```
STATUS status;  
INT16  timezone;  
UINT8  dst_enabled;  
  
status = SNTPC_Get_Timezone(&timezone, &dst_enabled);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Set\\_Timezone](#)

## SNTPC\_Get\_Time

This function returns the current time as determined from the most precise server. The current time is adjusted for timezone and DST.

### Usage

```
STATUS SNTPC_Get_Time (SNTPC_TIME *current_time);
```

### Arguments

- `current_time`

Pointer to a time structure that is filled out upon return.

[SNTPC\\_TIME](#) is defined in the [SNTP Client Data Structures](#) section of this chapter.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the `current_time` was retrieved.
- `SNTPC_OWP_NOT_RUN`  
The on-wire protocol has not been executed; `current_time` is invalid.
- `SNTP_NOT_SYNCED`  
The returned time has not been synchronized with a server.

### Example

```
STATUS      status;  
SNTPC_TIME sys_time;  
  
status = SNTPC_Get_Time(&sys_time);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Add\\_Server](#)

## SNTPC\_Get\_Time\_From\_Server

This function returns the current time as determined by the specified server. The returned current time is adjusted for timezone and DST.

### Usage

```
STATUS SNTPC_Get_Time_From_Server (SNTPC_TIME    *current_time,  
                                  SNTPC_SERVER  *server);
```

### Arguments

- `current_time`  
Pointer to a structure that is filled out upon return.  
[SNTPC\\_TIME](#) structure is defined in the [SNTP Client Data Structures](#) section of this chapter.
- `server`  
Pointer to a server structure that has been filled out.

### Return Values

- `NU_SUCCESS`  
The function completed successfully; the current time was retrieved.
- `SNTPC_NOT_SYNCED`  
The returned time has not been synchronized with a server.

### Example

```
STATUS      status;  
SNTPC_SERVER server;  
UINT8 serv_ip_addr[] = {192,168,1,1};  
SNTPC_TIME sys_time;  
  
/* Set up the server structure. */  
memcpy(&server.sntp_server_addr.id.is_ip_addrs, serv_ip_addr,  
       sizeof(serv_ip_addr));  
server.sntp_server_addr.family = NU_FAMILY_IP;  
server.sntp_server_addr.port = 123;  
server.sntp_poll_interval = 0;  
server.sntp_src_port = 123;  
  
status = SNTPC_Get_Time_From_Server(&sys_time, &server);
```

### Related Topics

[Nucleus SNTP Client - API Interface](#)

[SNTPC\\_Add\\_Server](#)

















# Chapter 15

## Simple Mail Transfer Protocol (SMTP)

---

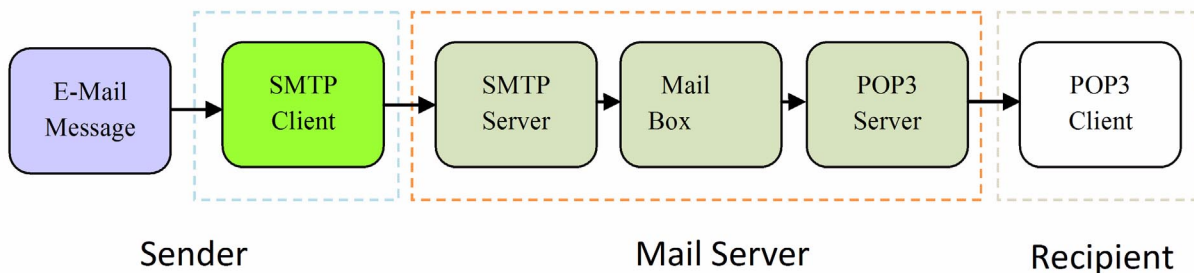
### SMTP Overview

Simple Mail Transfer Protocol is an IETF (Internet Engineering Task Force) defined network protocol which efficiently and reliably transfers Electronic Mail (E-Mail) from one user on a network to the mailbox of another user. SMTP is a very commonly used protocol on the Internet and is the backbone of all E-Mail services. [Figure 15-1](#) shows the different components of the protocol involved in transferring an E-Mail message from a sender to a recipient.

### SMTP Protocol

As shown in [Figure 15-1](#) the SMTP protocol is used to transfer an E-Mail message from a host to the mailbox of another user. The recipient uses another protocol known as Post Office Protocol 3 (POP3) to get the message from the mailbox on the server to his/her local computer. Our focus in this chapter is primarily on the SMTP Client component shown in the diagram.

**Figure 15-1. SMTP Overview**



### SMTP RFCs Overview

When SMTP was defined in RFC-821, it had many security issues. These issues were removed with the definition of Extended Simple Mail Transfer Protocol (ESMTP) in RFC-5321, which includes old SMTP with the addition of extensions. Extensions allow new methods to be added over generic SMTP implementation like authentication and data encryption.

### SMTP Protocol Extensions

The current version of the SMTP protocol is extensible. You can add extensions to the core protocol to add support for optional or advanced features such data encryption. There are

numerous extensions defined for the SMTP Protocol. Some of the more important extensions are described below:

- [SMTP Authentication](#)
- [SMTP Data Encryption](#)
- [SMTP Binary Messages](#)

## SMTP Authentication

The initial specification of the SMTP protocol had no support for user identity other than his E-Mail address. Authentication was included in RFC-4954 as an extension. It added Simple Authentication and Security Layer (SASL) support, which allows a user to authenticate himself/herself to the server before performing any transactions.

## SMTP Data Encryption

Without data encryption, E-Mail transactions are visible to anyone over the network. To add more security, Transport Layer Security (TLS) was introduced in RFC-2487. It allows a user to encrypt the connection before performing a transaction. Modern servers mostly use Secure Sockets Layer (SSL) for this purpose.

## SMTP Binary Messages

The legacy SMTP protocol was strictly a plain-text protocol. With the introduction of Multipurpose Internet Mail Extensions (MIME) users can send binary content in their E-Mail messages. The most commonly used extensions for this purpose are 8-Bit MIME (RFC-6152) and Binary MIME (RFC-3030).

# Nucleus SMTP Client Overview

Nucleus SMTP Client is an implementation of the SMTP protocol which allows you to send an E-Mail message. The Nucleus SMTP Client provides two types of user APIs. Firstly, it defines a set of "High-Level" APIs which provide an abstract interface for sending an E-Mail message. The "High-Level" interface is simple to use and hides all underlying details of the protocol. A "Low-Level" user API is also provided for advanced users who want more control of the communication with the server.

The following are the key features of the Nucleus SMTP Client:

- Supports Authentication and Encrypted connections to the server.
- Supports sending binary attachment(s) with an E-Mail.
- Allows sending an E-Mail message to multiple recipients.
- Supports 8-bit MIME.

- Provides a "High-Level" user-friendly interface and hides underlying protocol complexities.
- Provides a "Low-Level" user interface too, for advanced users.

Other important features are described below in more detail.

## Nucleus SMTP Authentication

Nucleus SMTP Client supports authentication and allows you to provide a user name and password to be sent for authentication. Supported methods include AUTH-PLAIN and AUTH-LOGIN defined in RFC-4954. The AUTH-LOGIN method is used by default and can be changed by using the [SMTP\\_Set\\_Extension\\_Settings](#) function.

## Nucleus SMTP Data Encryption

The Nucleus SMTP Client support both TLS and SSL methods for authentication. You must specify an SSL Certificate and provide its location through *.metadata* settings. The API automatically initializes an SSL context when SSL is required for a session. All E-Mail sessions use the same SSL Certificate which is specified in the configuration options.

## Attachments and Multi-Part Messages

The Nucleus Client API supports sending binary attachments in Base-64 format. A separate Message API is provided for this purpose which allows you to define multi-part messages. Currently it supports multiple text bodies and multiple binary/text attachments.

## Nucleus SMTP Supported RFCs

The following is the list of RFCs supported by SMTP Client API.

- RFC-2045 - Multipurpose Internet Mail Extensions (BASE64, MIME headers)
- RFC-2487 - SMTP Service Extension for Secures SMTP over TLS
- RFC-4954 - SMTP Service Extension for Authentication
- RFC-5321 - Simple Mail Transfer Protocol
- RFC-6409 - Message Submission for Mail (SMTP Extensions)

## Configuration Macros

The following configuration options for the SMTP Client are defined in the *os\include\networking\smtp\_client\_cfg.h* file.

## SMTP\_MAX\_CMD\_LINE

This macro defines the maximum length of an SMTP command. You can change this size to save memory. The default value is shown in [Table 15-1](#).

## SMTP\_MAX\_TXT\_LINE

This macro defines the maximum length of text line in a message. The default value is shown in [Table 15-1](#).

## SMTP\_TIMEOUT

This macro defines the connection timeout for receiving a reply from the server. It is only applicable for non-encrypted connections. The default value is 10 seconds.

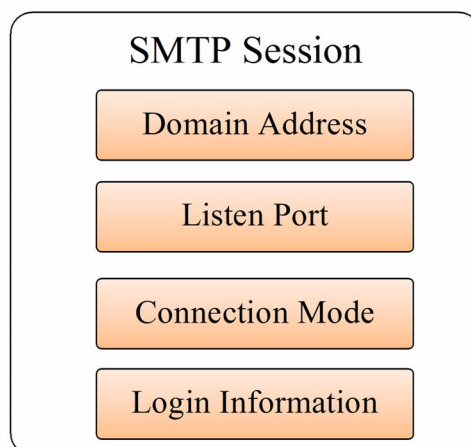
# SMTP Client Theory of Operation

The high-level API of the SMTP Client primarily uses two opaque objects for providing E-Mail services to the user. These two objects are described in detail in the [SMTP Session](#) and [SMTP Message](#) sections.

## SMTP Session

The SMTP Session object stores the SMTP server configuration and maintains the internal state of the client. It is initialized using the [NU\\_SMTP\\_Session\\_Init](#) function. Data such as the server address, port number, authentication method and connection type/state are stored in this object. All API functions use this object for communicating with the server. The SMTP Session object is briefly described in [Figure 15-2](#).

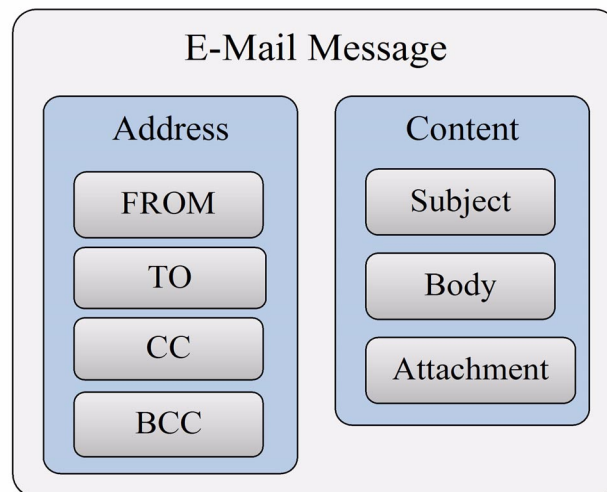
**Figure 15-2. SMTP\_SESSION**



## SMTP Message

The SMTP Message object stores the complete E-Mail message and its envelope. The message is composed of its subject, body and attachments whereas its envelope contains information such as the sender and recipients. This object is initialized using the [NU\\_SMTP\\_Msg\\_Init](#) function and is populated using various message construction API functions such as [NU\\_SMTP\\_Msg\\_Add\\_Attachment](#), [NU\\_SMTP\\_Msg\\_Add\\_Body](#), [NU\\_SMTP\\_Msg\\_Add\\_Recipient](#), [NU\\_SMTP\\_Msg\\_Set\\_Subject](#) and [NU\\_SMTP\\_Msg\\_Set\\_Sender](#). The SMTP Message object is finally sent to the SMTP Server using the [NU\\_SMTP\\_Send\\_Message](#) API function. The SMTP Message object is described in [Figure 15-3](#).

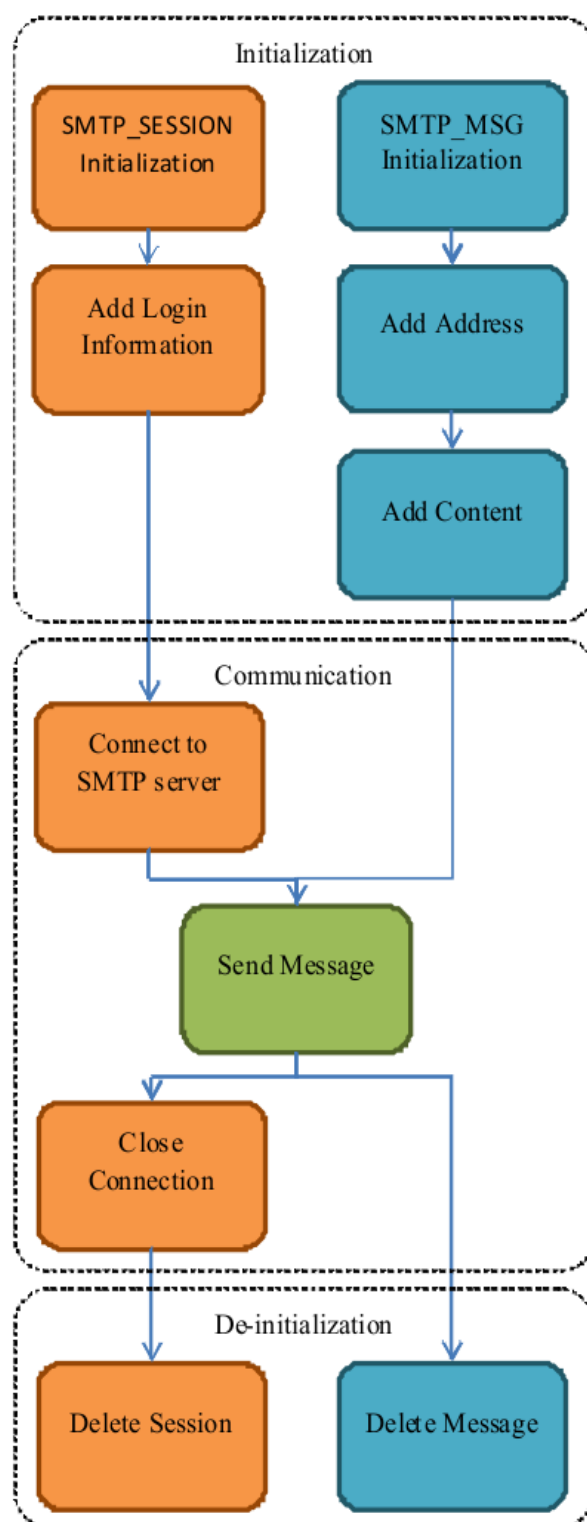
**Figure 15-3. SMTP\_MSG Object**



## High-Level API Control Flow

The SMTP library is designed to provide a user-friendly interface to send an E-Mail with minimal knowledge of underlying protocols. [Figure 15-4](#) gives an overview of how the high-level SMTP API is used in three phases: initialization, communication and de-initialization.

**Figure 15-4. High-Level API Control Flow**

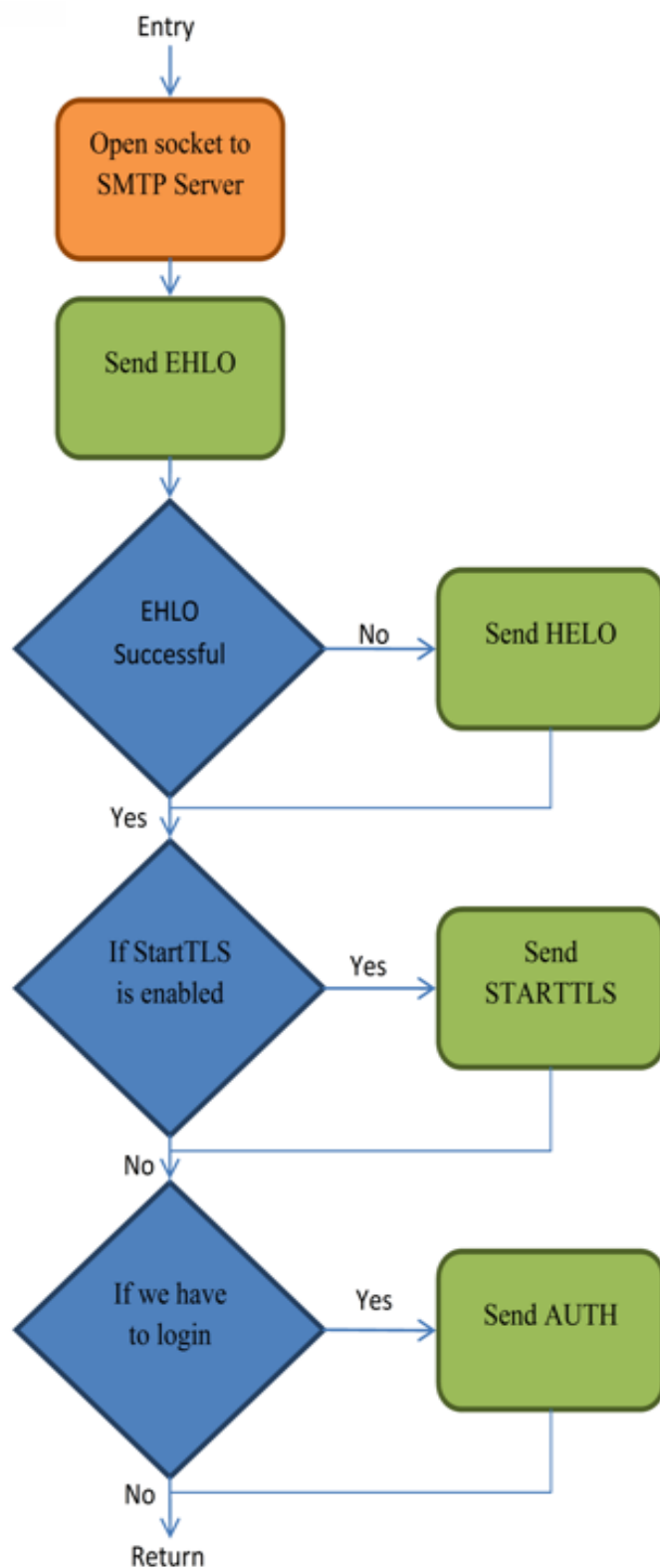




## Session Connect Sequence

Each API of the high-level SMTP interface may map to multiple underlying primitive SMTP operations. [Figure 15-5](#) describes the internal operations performed by the [NU\\_SMTP\\_Session\\_Connect](#) API function. This sequence of operations is in accordance with the SMTP RFC. An advanced user may use the low-level SMTP API to customize or modify this default sequence of operations.

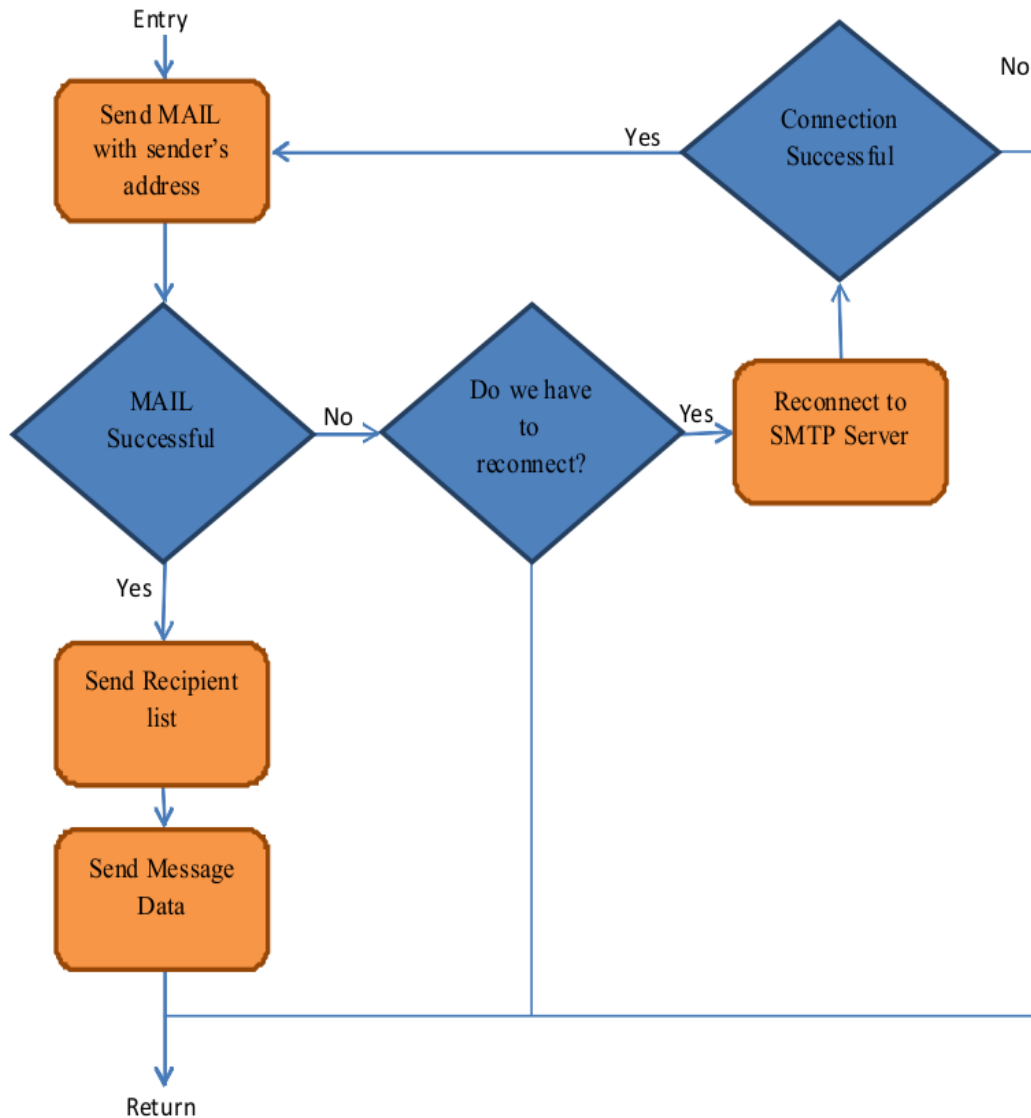
**Figure 15-5. Session Connect Sequence**



## Send Message Sequence

Figure 15-6 in this section describes the internal sequence of operations performed by the `NU_SMTP_Send_Message` API function.

**Figure 15-6. Send Message Sequence**



## SMTP Sizes and Limits

SMTP sizes are defined in RFC-5321 Section 4.5.3.1. According to this section these limits must be kept in mind when setting an address, subject or text content of message.

**Table 15-1. Sizes and Limits**

Section	Size limit in Bytes
Address	254
Text line in a message	1000
Command	512
Subject	985
Maximum message size	Depends on the SMTP Server

## SMTP Client High-Level API Functions

This section describes the high level APIs provided for the creation of sessions, messages, and sending an E-mail.

- [NU\\_SMTP\\_Msg\\_Add\\_Attachment](#)
- [NU\\_SMTP\\_Msg\\_Add\\_Body](#)
- [NU\\_SMTP\\_Msg\\_Add\\_Recipient](#)
- [NU\\_SMTP\\_Msg\\_Delete](#)
- [NU\\_SMTP\\_Msg\\_Init](#)
- [NU\\_SMTP\\_Msg\\_Set\\_Subject](#)
- [NU\\_SMTP\\_Msg\\_Set\\_Sender](#)
- [NU\\_SMTP\\_Send\\_Message](#)
- [NU\\_SMTP\\_Session\\_Connect](#)
- [NU\\_SMTP\\_Session\\_Disconnect](#)
- [NU\\_SMTP\\_Session\\_Delete](#)
- [NU\\_SMTP\\_Session\\_Init](#)
- [NU\\_SMTP\\_Session\\_Set\\_Login](#)

## NU\_SMTP\_Msg\_Add\_Attachment

This function adds an attachment to a message.

### Usage

```
STATUS NU_SMTP_Msg_Add_Attachment (SMTP_MSG *message,  
                                   CHAR      *attachment,  
                                   CHAR      *name,  
                                   INT32    len,  
                                   INT16    type);
```

### Arguments

- **message**  
Pointer to SMTP\_MSG object that was initialized earlier.
- **attachment**  
Pointer to the contents of the attachment.
- **name**  
Pointer to the filename that is sent for this attachment.
- **len**  
Length of the data that is copied from the address pointed by attachment.
- **type**  
The format in which the attachment is sent. This can be text or binary:  
SMTP\_ATT\_TXT  
SMTP\_ATT\_BIN

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the attachment is added to the message object.
- **NU\_INVALID\_PARM**  
An invalid parameter was passed.

### Description

Attachments can be either binary or plain-text data. You can choose the type of attachment according to your requirements.

SMTP\_ATT\_TXT specifies that data is in plain-text ASCII format and can be used to send plain text files. SMTP\_ATT\_BIN specifies that data will be sent in binary format. This API will internally convert binary format into a Base-64 encoded format which usually increases the attachment size by about 30%.

### **Caution**



The receiver might not receive an exact copy of the data when using text format if it contains lines longer than the maximum limit supported by the server.

---

For limits and sizes please refer to [SMTP Sizes and Limits](#) section of this chapter.

### **Example**

```
SMTP_MSG *message;
CHAR *attachment = "This is a text attachment.";

/* Initialize the message object. */
message = NU_SMTP_Msg_Init ();

/* Add an attachment to the message object. */
NU_SMTP_Msg_Add_Attachment (message,
                             attachment,
                             "text_attachment.txt",
                             strlen (attachment),
                             SMTP_ATT_TXT);
```

### **Related Topics**

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Add\\_Body](#)

## NU\_SMTP\_Msg\_Add\_Body

This function adds a body to a message.

### Usage

```
STATUS NU_SMTP_Msg_Add_Body (SMTP_MSG *message,  
                             CHAR      *body_txt);
```

### Arguments

- `message`  
SMTP\_MSG object that was initialized earlier.
- `body_txt`  
Array containing body content.

### Return Values

- `NU_SUCCESS`  
Function completed successfully, the body was added to message object.
- `NU_INVALID_PARM`  
An invalid parameter was passed.

### Description

The body is sent in text format so you can only use plain-text ASCII characters. There can be multiple bodies in a message like paragraphs.

For limits and sizes please refer to [SMTP Sizes and Limits](#) section in this chapter.

### Example

```
SMTP_MSG *message;  
CHAR *body = "This is body of message."  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();  
  
/* Add a body to the message object. */  
NU_SMTP_Msg_Add_Body (message, body);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Add\\_Attachment](#)

## NU\_SMTP\_Msg\_Add\_Recipient

This function adds a recipient to a message.

### Usage

```
STATUS NU_SMTP_Msg_Add_Recipient (SMTP_MSG *message,  
                                CHAR      *recpt,  
                                INT16     recpt_type);
```

### Arguments

- **message**  
SMTP\_MSG object that was initialized earlier.
- **recpt**  
Array containing the recipient address.
- **recpt\_type**

Valid recpt\_type options are:

SMTP\_MSG\_TO

This adds the recipient to the “To” field of the outgoing E-mail.

SMTP\_MSG\_CC

This adds the recipient to the “CC” (Carbon Copy) field of the outgoing E-mail.

SMTP\_MSG\_BCC

This adds the recipient to the “BCC” (Blind Carbon Copy) field of the outgoing E-mail.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the recipient was added to the message object.
- **NU\_INVALID\_PARM**  
An input parameter was invalid.

### Description

The different types of recipients are described in the recpt\_type argument. For sizes and limits please refer to the [SMTP Sizes and Limits](#) section in this chapter.

### Example

```
SMTP_MSG *message;  
CHAR *recpt = "dummy@example.com";  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();
```



```
/* Add the recipient address. */  
NU_SMTP_Msg_Add_Recipient (message, recpt, SMTP_MSG_TO);
```

## Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Add\\_Body](#)

## NU\_SMTP\_Msg\_Delete

This function deletes a previously initialized SMTP\_MSG object.

### Usage

```
STATUS NU_SMTP_Msg_Delete (SMTP_MSG *message);
```

### Arguments

- **message**  
SMTP\_MSG object which was initialized earlier and which needs to be deleted.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the object was de-allocated.
- **NU\_INVALID\_PARM**  
An input parameter was invalid.

### Description

This function also frees any resources associated with the SMTP\_MSG object, which includes addresses, bodies and attachments added to this message object. This API must be called on an SMTP\_MSG object after the object is no longer required.

### Example

```
SMTP_MSG *message;  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();  
.  
.  
.  
/* Delete the message object. */  
NU_SMTP_Msg_Delete (message);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Init](#)

## NU\_SMTP\_Msg\_Init

This function initializes an SMTP\_MSG object and returns its pointer.

### Usage

```
SMTP_MSG *NU_SMTP_Msg_Init (VOID);
```

### Arguments

- None

### Return Values

- SMTP\_MSG  
Function completed successfully, and returns a pointer to the object initialized.
- NU\_NULL  
The API was unable to allocate memory for this object.

### Description

This object can be further populated using other APIs (see the "Related Topics" section of this API).

### Example

```
SMTP_MSG *message;  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();
```

### Related Topics

<a href="#">SMTP Client High-Level API Functions</a>	<a href="#">NU_SMTP_Msg_Delete</a>
<a href="#">NU_SMTP_Msg_Add_Attachment</a>	<a href="#">NU_SMTP_Msg_Add_Body</a>
<a href="#">NU_SMTP_Msg_Add_Recipient</a>	<a href="#">NU_SMTP_Msg_Set_Sender</a>
<a href="#">NU_SMTP_Send_Message</a>	<a href="#">NU_SMTP_Msg_Delete</a>

## NU\_SMTP\_Msg\_Set\_Subject

This function sets the subject of the E-Mail message.

### Usage

```
STATUS NU_SMTP_Msg_Set_Subject (SMTP_MSG *message,  
                                CHAR      *subject);
```

### Arguments

- **message**  
SMTP\_MSG object that was initialized earlier.
- **subject**  
Array containing the subject of the message.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the message subject was set.
- **NU\_INVALID\_PARM**  
An input parameter was invalid.

### Description

A message object can have only one subject. If multiple calls are performed on a message object, only the last subject will be applied.

For limits and sizes please refer to [SMTP Sizes and Limits](#) section of this chapter.

### Example

```
SMTP_MSG *message;  
CHAR *subject = "Test Email";  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();  
  
/* Set the subject. */  
NU_SMTP_Msg_Set_Subject (message, subject);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Set\\_Sender](#)

## NU\_SMTP\_Msg\_Set\_Sender

This function sets the address of the sender in the message object.

### Usage

```
STATUS NU_SMTP_Msg_Set_Sender (SMTP_MSG *message,  
                               CHAR      *sender);
```

### Arguments

- **message**  
SMTP\_MSG object that was initialized earlier.
- **sender**  
A null-terminated string containing the address of the sender.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the address of the sender was set in the message object.
- **NU\_INVALID\_PARM**  
An input parameter was invalid.

### Description

There can be only one address of the sender, if multiple calls are performed on a single message object, the last address will be used when sending the message. This address must be consistent with the session settings to avoid problems with the server.

For limits and sizes please refer to [SMTP Sizes and Limits](#) section of this chapter.

### Example

```
SMTP_MSG *message;  
CHAR *sender = "dummy@example.com";  
  
/* Initialize the message object. */  
message = NU_SMTP_Msg_Init ();  
  
/* Set the sender address. */  
NU_SMTP_Msg_Set_Sender (message, sender);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Set\\_Subject](#)

## NU\_SMTP\_Send\_Message

This function delivers an E-Mail message to the server.

### Usage

```
STATUS NU_SMTP_Send_Message (SMTP_SESSION *session,  
                             SMTP_MSG      *message);
```

### Arguments

- session  
SMTP\_SESSION object that was initialized earlier.
- message  
SMTP\_MSG object that was initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the message was sent.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Description

The message was earlier constructed using other API functions (see the "Related Topics" section of this API). Both message and session objects must be populated correctly to send a message successfully.

### Example

```
SMTP_SESSION *session;  
SMTP_MSG *message;  
  
/* Initialize the message and session objects. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
message = NU_SMTP_Msg_Init();  
  
/* Add To and From addresses in the message. */  
NU_SMTP_Msg_Set_Sender (message, "user1@example.com");  
NU_SMTP_Msg_Add_Recipient (message, "user2@example.com", SMTP_MSG_TO);  
  
/* Set the subject of the message. */  
NU_SMTP_Msg_Set_Subject (message, "This is a Subject");  
  
/* Add a body to the message. */  
NU_SMTP_Msg_Add_Body (message, body);  
  
/* Connect and send the message. */  
NU_SMTP_Session_Connect (session);  
NU_SMTP_Send_Message (session, message);
```

```
/* Disconnect and send the message. */  
NU_SMTP_Session_Disconnect (session);  
NU_SMTP_Session_Delete (session);  
NU_SMTP_Msg_Delete (message);
```

## Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Msg\\_Init](#)

[NU\\_SMTP\\_Session\\_Init](#)

## NU\_SMTP\_Session\_Connect

This function connects and logs into the SMTP server according to session settings.

### Usage

```
STATUS NU_SMTP_Session_Connect (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object that was initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the connection to the server was accomplished.
- SMTP\_TLS\_ERROR  
The user is trying to use Start-TLS for encryption and the server does not support this feature.
- SMTP\_ATUH\_ERROR  
The user is required to send login information but no login information was found.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Description

Call the [NU\\_SMTP\\_Session\\_Init](#) and [NU\\_SMTP\\_Session\\_Set\\_Login](#) API functions first to construct the session object before calling this API function.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

/* Connect to the SMTP server. */
NU_SMTP_Session_Connect (session);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Session\\_Disconnect](#)

[NU\\_SMTP\\_Send\\_Message](#)



## NU\_SMTP\_Session\_Disconnect

This function closes a previously opened connection to an SMTP server.

### Usage

```
STATUS NU_SMTP_Session_Disconnect (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object that was initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the session was disconnected.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

/* Connect to the SMTP server. */
NU_SMTP_Session_Connect (session);
.
.
.
/* Disconnect form the SMTP server. */
NU_SMTP_Session_Disconnect (session);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Session\\_Connect](#)

[NU\\_SMTP\\_Send\\_Message](#)

## NU\_SMTP\_Session\_Delete

This function deletes a session object and frees all associated resources.

### Usage

```
STATUS NU_SMTP_Session_Delete (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object that was initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the session object was de-allocated.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Description

This API must be called on a session object when it is no longer required.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);
.
.
.
/* Delete the session object. */
NU_SMTP_Session_Delete (session);
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Session\\_Init](#)

[NU\\_SMTP\\_Send\\_Message](#)

[NU\\_SMTP\\_Session\\_Set\\_Login](#)

## NU\_SMTP\_Session\_Init

This function initializes an SMTP\_SESSION object according to its settings and returns its pointer.

### Usage

```
SMTP_SESSION *NU_SMTP_Session_Init (CHAR    *address,  
                                     UINT16  port,  
                                     INT16   connection_mode);
```

### Arguments

- **address**  
A character array representing the SMTP server address. This can be a domain address, IPv4 address string, or IPv6 address string. In case of a domain address this function will try to resolve the address.
- **port**  
Port on which the SMTP server is listening for SMTP connections.
- **connection\_mode**  
Connection mode that will be used when connecting to the SMTP server. You can specify one of the following connection modes for a session.

SMTP\_OPEN

If the server connection is not encrypted.

SMTP\_SSL

For using SSL to connect to the SMTP server.

SMTP\_TLS

For using Start-TLS when encrypting the connection.

### Return Values

- **SMTP\_SESSION**  
The function completed successfully, the object was initialized.
- **NU\_NULL**  
The API was unable to allocate memory for this object.

### Example

```
SMTP_SESSION *session;  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);
```

## Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Session\\_Delete](#)

[NU\\_SMTP\\_Session\\_Set\\_Login](#)

[NU\\_SMTP\\_Send\\_Message](#)

## NU\_SMTP\_Session\_Set\_Login

This function sets the login information in a session object.

### Usage

```
STATUS NU_SMTP_Session_Set_Login (SMTP_SESSION *session,  
                                CHAR            *username,  
                                CHAR            *password);
```

### Arguments

- session  
SMTP\_SESSION object that was initialized earlier.
- username  
Pointer to the user name that will be used to login into the server.
- password  
Pointer to the corresponding user password.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the login information was added.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
  
/* Set the login information. */  
NU_SMTP_Session_Set_Login(session, "dummy@example.com", "my-secret");
```

### Related Topics

[SMTP Client High-Level API Functions](#)

[NU\\_SMTP\\_Send\\_Message](#)

[NU\\_SMTP\\_Send\\_Message](#)

[NU\\_SMTP\\_Session\\_Init](#)

## SMTP Client Low-Level API Functions

This section describes the low-level APIs of the SMTP Client. These APIs are for advanced users who may have special needs for communicating with the SMTP Server.

Even while using low-level APIs, the following high-level API functions must be called to manage the SMTP Session:

- [NU\\_SMTP\\_Session\\_Init](#)
- [NU\\_SMTP\\_Session\\_Set\\_Login](#)
- [NU\\_SMTP\\_Session\\_Disconnect](#)
- [NU\\_SMTP\\_Session\\_Delete](#)

Once a session has been established, you can use the following low-level API functions to communicate with the SMTP Server and for sending E-Mail messages. All of these functions are described in detail in the following sections:

- [NU\\_SMTP\\_Auth](#)
- [NU\\_SMTP\\_Base\\_Cmd](#)
- [NU\\_SMTP\\_Close](#)
- [NU\\_SMTP\\_Data](#)
- [NU\\_SMTP\\_Ehlo](#)
- [NU\\_SMTP\\_Helo](#)
- [NU\\_SMTP\\_Mail](#)
- [NU\\_SMTP\\_Msg](#)
- [NU\\_SMTP\\_Quit](#)
- [NU\\_SMTP\\_Rcpt](#)
- [NU\\_SMTP\\_Rset](#)
- [NU\\_SMTP\\_StartTLS](#)
- [SMTP\\_Client\\_Init](#)
- [SMTP\\_Base64\\_Encode](#)
- [SMTP\\_Set\\_Extension\\_Settings](#)

## NU\_SMTP\_Auth

This function sends login information to the server according to authentication settings.

### Usage

```
STATUS NU_SMTP_Auth (SMTP_SESSION *session,  
                    CHAR           *username,  
                    CHAR           *password);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- username  
Pointer to the user name to be sent to the server.
- password  
Pointer to password associated with the user name.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the user is logged in.

### Example

```
SMTP_SESSION *session;  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
  
. . .  
/* Try to log into the server. */  
NU_SMTP_Auth (session, "dummy@example.com", "dummy-password");
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Base\\_Cmd](#)

## NU\_SMTP\_Base\_Cmd

This is the base function for all commands supported by SMTP Client API.

### Usage

```
STATUS NU_SMTP_Base_Cmd (SMTP_SESSION *session,  
                        INT16         command,  
                        CHAR          *argument,  
                        INT16         *ok_rep,  
                        CHAR          *reply);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- command  
The command to be sent to the server. You can provide one of the following commands:  
SMTP\_DATA\_CMD  
SMTP\_EXPN\_CMD  
SMTP\_HELP\_CMD  
SMTP\_MAIL\_CMD  
SMTP\_NOOP\_CMD  
SMTP\_RCPT\_CMD  
SMTP\_VRFY\_CMD
- argument  
An extra argument to the command. Set this to NU\_NULL if no argument is required.
- ok\_rep  
A null-terminated array of status codes which the server may reply with if the command was successful. You must specify at least one status code.
- reply  
If not NU\_NULL then the reply of the server will be copied into this buffer. The minimum length of this buffer is SMTP\_MAX\_CMD\_LINE. For default size, see Table [15-1](#).

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- SMTP\_UNKNOWN\_COMMAND  
The command was not found.



- **NU\_INVALID\_PARM**  
An input parameter is invalid.

### Example

```
SMTP_SESSION *session;
CHAR reply [100];
INT16 ok_rep [2] = {250, 0};

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

/* Connect to the SMTP server. */
NU_SMTP_Session_Connect (session);

/* Send the NOOP command. */
NU_SMTP_Base_Cmd (session, SMTP_NOOP_CMD, NU_NULL, ok_rep, reply);

/* Print the reply from server. */
printf("%s", reply);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Auth](#)

## NU\_SMTP\_Close

This function closes a connection to the SMTP server.

### Usage

```
STATUS NU_SMTP_Close (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the socket was closed.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.

/* Connect to the SMTP server. */
NU_SMTP_Session_Connect (session);

.
.
.

/* Disconnect from the server. */
NU_SMTP_Close (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Data](#)

## NU\_SMTP\_Data

This function sends the DATA command to the SMTP server to prepare the server for receiving the data. To send the actual data to the server, you must call the [NU\\_SMTP\\_Msg](#) API after calling [NU\\_SMTP\\_Data](#).

### Usage

```
STATUS NU_SMTP_Data (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send the DATA command. */
NU_SMTP_Data (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Ehlo](#)

## NU\_SMTP\_Ehlo

This function sends the EHLO (Extended Hello) command to the server.

### Usage

```
STATUS NU_SMTP_Ehlo (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Description

This is usually the first command sent to the server to query the server configuration. By default the IP address of the client is sent as a domain address.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send the EHLO command. */
NU_SMTP_Ehlo (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Helo](#)

## NU\_SMTP\_Helo

This function sends the HELO (HELLO) command to the server. By default the IP address of the client is sent as the domain address.

### Usage

```
STATUS NU_SMTP_Helo (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send HELO command. */
NU_SMTP_Helo (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Ehlo](#)

## NU\_SMTP\_Mail

This function sends the MAIL command to the server. The address of the sender is provided using the “from” parameter.

### Usage

```
STATUS NU_SMTP_Mail (SMTP_SESSION *session,  
                    CHAR           *from);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- from  
The address of the sender as a null-terminated string of characters.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
  
. . .  
  
/* Send the MAIL command. */  
NU_SMTP_Mail (session, "dummy@example.com");
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Msg](#)

## NU\_SMTP\_Msg

This function sends message data to the server.

### Usage

```
STATUS NU_SMTP_Msg (SMTP_SESSION *session,  
                   CHAR           *data,  
                   INT32          length);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- data  
Pointer to message data.
- length  
The length of data in bytes.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Description

You must not terminate this message data with a <CRLF>.<CRLF> sequence. The function will send the <CRLF>.<CRLF> terminating sequence at the end of the message data.

### Example

```
SMTP_SESSION *session;  
CHAR *message =  
"FROM: <dummy@example.com>\r\n"  
"TO: <dummy2@example.com>\r\n"  
"SUBJECT: Example Message\r\n"  
"This a small message";  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
  
.  
.  
.  
/* Send the DATA command. */  
NU_SMTP_Data (session);  
  
/* Send the message data. */  
NU_SMTP_Msg (session, message, strlen (message));
```

## Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Mail](#)



## NU\_SMTP\_Quit

This function sends the QUIT command to the server.

### Usage

```
STATUS NU_SMTP_Quit (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send the QUIT command. */
NU_SMTP_Quit (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Rset](#)

## NU\_SMTP\_Rcpt

This function sends the RCPT (Recipient) command to server. The address of the recipient is provided using the "to" parameter.

### Usage

```
STATUS NU_SMTP_Rcpt (SMTP_SESSION *session,  
                    CHAR           *to);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- to  
The address of the recipient.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;  
  
/* Initialize the session object. */  
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);  
  
.  
.  
.  
  
/* Send the RCPT command. */  
NU_SMTP_Rcpt (session, "dummy@example.com");
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Rset](#)

## NU\_SMTP\_Rset

This function sends the RSET (RESET) command to the server.

### Usage

```
STATUS NU_SMTP_Rset (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send the RSET command. */
NU_SMTP_Rset (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[NU\\_SMTP\\_Rcpt](#)

## NU\_SMTP\_StartTLS

This function sends the STARTTLS command to the server. The connection will be encrypted after this command is successfully completed.

### Usage

```
STATUS NU_SMTP_StartTLS (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully.
- NU\_INVALID\_PARM  
An input parameter was invalid.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Send the STARTTLS command. */
NU_SMTP_StartTLS (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[SMTP\\_Client\\_Init](#)

## SMTP\_Client\_Init

This function connects to the SMTP server. If an SSL connection is selected in the session settings, then an SSL socket is connected.

### Usage

```
STATUS SMTP_Client_Init (SMTP_SESSION *session);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the client is connected to the server.
- NU\_INVALID\_PARM  
The session object is NU\_NULL.
- SMTP\_CONNECT\_FAILED  
The connection to the server was not successful.
- SMTP\_SSL\_INIT\_ERROR  
Unable to initialize SSL.
- SMTP\_RECEIVE\_FAILED  
Unable to receive a welcome message from the server.

### Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_Smtp_Session_Init ("smtp.example.com", 25, SMTP_OPEN);

.
.
.
/* Connect to the SMTP server. */
SMTP_Client_Init (session);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[SMTP\\_Base64\\_Encode](#)

## SMTP\_Base64\_Encode

This function is a helper function used to encode data into Base-64 encoding. You must deallocate the result buffer after it has been used.

### Usage

```
STATUS SMTP_Base64_Encode (CHAR    *string,  
                           CHAR    **result,  
                           UINT32  length);
```

### Arguments

- **string**  
Pointer to the data to be encoded.
- **result**  
Return pointer to the encoded data contained in the dynamically allocated buffer which must be freed by the caller.
- **length**  
Length of the data to be encoded.

### Example

```
CHAR data[] = "My dummy data";  
CHAR *result;  
  
/* Convert the string to Base64 format. */  
SMTP_Base64_Encode (data, &result, strlen (data));  
printf ("%s\r\n", result);  
  
/* Deallocate the return buffer. */  
NU_Deallocate_Memory (result);
```

### Related Topics

[SMTP Client Low-Level API Functions](#)

[SMTP\\_Set\\_Extension\\_Settings](#)

## SMTP\_Set\_Extension\_Settings

This function sets the extension settings. Currently only Authentication (RFC-4954) and Start-TLS (RFC-2487) are available.

### Usage

```
STATUS SMTP_Set_Extension_Settings(SMTP_SESSION *session,  
                                  INT16          extension,  
                                  INT16          setting);
```

### Arguments

- session  
SMTP\_SESSION object initialized earlier.
- extension  
Extension identifier can be  
SMTP\_AUTH\_EXTEN  
For the “Authentication” extension.  
SMTP\_TLS\_EXTEN  
For the “Start-TLS” extension.
- setting  
Setting for that extension. Table 15-2 shows valid options for the two extensions.

**Table 15-2. Setting Options**

SMTP_AUTH_EXTEN	Description	SMTP_TLS_EXTEN	Description
SMTP_AUTH_DISABLED	disable authentication	SMTP_TLS_DISABLED	disable Start-TLS
SMTP_AUTH_PLAIN	use plain-text authentication	SMTP_TLS_ENABLED	enable Start-TLS
SMTP_AUTH_LOGIN	use login authentication		

### Return Values

- NU\_SUCCESS  
Function completed the request successfully.
- SMTP\_SETTING\_ERROR  
An error occurred while performing the request.
- SMTP\_INVALID\_PARM  
An input argument is invalid.

## Example

```
SMTP_SESSION *session;

/* Initialize the session object. */
session = NU_SMTP_Session_Init("smtp.example.com", 25, SMTP_OPEN);

/* Use login authentication. */
SMTP_Set_Extension_Settings (session, SMTP_AUTH_EXTEN, SMTP_AUTH_LOGIN);
```

## Related Topics

[SMTP Client Low-Level API Functions](#)

[SMTP\\_Client\\_Init](#)

































































## JSON Generator Overview

JavaScript Object Notation (JSON) is a data interchange format which is frequently used in Web Applications, and on the Internet, to transfer structured data in a machine independent format.

## Nucleus JSON Generator Implementation

The Nucleus JSON Generator allows you to create a JSON Document. A Nucleus JSON Document is being composed of two types:

- Tokens

Tokens are used to contain data, and can be:

- Objects and Arrays
- Values

A value is a primitive type, in this case defined as either:

- Boolean, Integer, String, Null, or Float

---

**Note**

Note that floats are represented using strings to ensure there are no rounding issues.

---

## JSON Generator Data Structures and Constants

In order to use the Nucleus JSON Generator API, you need to be aware of the following structure and defines:

The member of the JSON Generator structure is defined in [Table 16-1](#):

**Table 16-1. JSON Generator Structure**

Member	Description
JSON_GEN_HANDLE	Public access to JSON Generator's context via an opaque pointer

The JSON Generator constants are defined in [Table 16-2](#):

**Table 16-2. JSON Generator Constants**

Name	Description
JSON_TOKEN_TYPE_OBJECT	JSON token object type
JSON_TOKEN_TYPE_ARRAY	JSON token array type
JSON_MAX_DEPTH_LEVEL	Maximum depth of nesting in the JSON structure
JSON_MAX_INTEGER_LENGTH	Maximum length of a 64-bit integer including the sign and the null terminator

## Interface in Action

Following is an example of a JSON Document which we need to generate programmatically.

### Sample JSON Document

```
{
  "person": {
    "first_name": "Bob",
    "last_name": "Smith",
    "dog": {
      "breed": "Boxer",
      "age": 5.6
    },
    "TV Shows": [
      "Buffy",
      "Farscape"
    ],
    "cars": [
      {
        "4Runner": {
          "color": "teal",
          "cylinders": 6,
          "year": 2008,
          "full_paid_for": true,
          "amount_owed": null
        }
      },
      {
        "Tahoe": {
          "color": "green",
          "cylinders": 8,
          "year": 2005,
          "full_paid_for": false,
          "amount_owed": 10000
        }
      }
    ]
  }
}
```

## Sample JSON Generator Code

This example does not perform any type of error checking.

```
INT32          jint = 0;
BOOLEAN        jbool = NU_FALSE;
JSON_GEN_HANDLE *handle;
#define BUFFER_SIZE 512

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* Start the JSON String. */
NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                              JSON_TOKEN_TYPE_OBJECT);

/* Start the Person Object.*/
NU_JSON_Generator_Start-Token(&handle, "person",
                              JSON_TOKEN_TYPE_OBJECT);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "first_name");
NU_JSON_Generator_Add_String(&handle, "Bob", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "last_name");
NU_JSON_Generator_Add_String(&handle, "Smith", 0);

/* Add a Dog Object. */
NU_JSON_Generator_Start-Token(&handle, "dog",
                              JSON_TOKEN_TYPE_OBJECT);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "breed");
NU_JSON_Generator_Add_String(&handle, "Boxer", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "age");
NU_JSON_Generator_Add_Float(&handle, "5.6");

/* End of Dog Object */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Add a TV Show Array. */
NU_JSON_Generator_Start-Token(&handle, "TV Shows",
                              JSON_TOKEN_TYPE_ARRAY);

/* Add values to our Array of TV Shows. */
NU_JSON_Generator_Add_String(&handle, "Buffy", 0);
NU_JSON_Generator_Add_String(&handle, "Farscape", 0);

/* End of TV Show Array */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_ARRAY);

/* Start Cars Array. */
NU_JSON_Generator_Start-Token(&handle, "Cars",
                              JSON_TOKEN_TYPE_ARRAY);
```

```
/* Start Car Object => 4Runner. */
NU_JSON_Generator_Start-Token(&handle, "4Runner",
                               JSON_TOKEN_TYPE_OBJECT);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "color");
NU_JSON_Generator_Add_String(&handle, "teal", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "cylinders");
NU_JSON_Generator_Add_UInt(&handle, 6);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "year");
NU_JSON_Generator_Add_UInt(&handle, 2008);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "full_paid_for");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "amount_owed");
NU_JSON_Generator_Add_Null(&handle);

/* Close a 4Runner Car Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT)

/* Start a Car Object => Tahoe. */
NU_JSON_Generator_Start-Token(&handle,
                              "Tahoe", JSON_TOKEN_TYPE_OBJECT);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "color");
NU_JSON_Generator_Add_String(&handle, "green", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "cylinders");
NU_JSON_Generator_Add_UInt(&handle, 8);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "year");
NU_JSON_Generator_Add_UInt(&handle, 2005);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "full_paid_for");
NU_JSON_Generator_Add_Boolean(&handle, NU_FALSE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "amount_owed");
NU_JSON_Generator_Add_Int(&handle, 10000);

/* Close the Tahoe Car Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Close Cars Array. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_ARRAY);
```



```
/* Close the Person Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON String. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);
NU_JSON_Generator_Destroy(&handle);
```

## JSON Generator APIs

This section describes the Nucleus JSON Generator APIs. In order to make use of these APIs, you need to include the following header file in your application as shown below:

```
#include "networking/nu_json.h"
```

- [NU\\_JSON\\_Generator\\_Create](#)
- [NU\\_JSON\\_Generator\\_Destroy](#)
- [NU\\_JSON\\_Generator\\_Start-Token](#)
- [NU\\_JSON\\_Generator\\_End-Token](#)
- [NU\\_JSON\\_Generator\\_Add\\_Name](#)
- [NU\\_JSON\\_Generator\\_Add\\_String](#)
- [NU\\_JSON\\_Generator\\_Add\\_Boolean](#)
- [NU\\_JSON\\_Generator\\_Add\\_Int](#)
- [NU\\_JSON\\_Generator\\_Add\\_UInt](#)
- [NU\\_JSON\\_Generator\\_Add\\_Float](#)
- [NU\\_JSON\\_Generator\\_Add\\_Null](#)
- [NU\\_JSON\\_Generator\\_Get\\_Buffer](#)
- [NU\\_JSON\\_Generator\\_Clear\\_Buffer](#)

## NU\_JSON\_Generator\_Create

Creates a new instance of a JSON Generator. The JSON Generator is used to construct a JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Create(UINT32          buffer_size,  
                                JSON_GEN_HANDLE *handle);
```

### Arguments

- **buffer\_size**  
Buffer size to be allocated by the generator. This buffer holds the generated JSON Document. If the buffer is smaller than the complete size of the JSON Document being generated, trying to add data to the buffer will return a `NUF_JSON_BUFFER_FULL` error. Then you must call [NU\\_JSON\\_Generator\\_Get\\_Buffer](#) to obtain the filled-up buffer and process it, followed by a call to [NU\\_JSON\\_Generator\\_Clear\\_Buffer](#) to clear the buffer so that more data can be added to it.
- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#), used when calling the Generators's APIs.

### Return Values

- `NU_SUCCESS`  
The function completed successfully, the JSON Generator was created.
- `< 0`  
Error codes.

### Examples

```
JSON_GEN_HANDLE    *handle;  
STATUS             status;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator. */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
  
/* JSON Generator was created. */  
if(status == NU_SUCCESS){  
    /* ... */  
}  
/* There was an error. */  
else  
{  
    /* Report error. */  
}
```

### Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Destroy](#)

## NU\_JSON\_Generator\_Destroy

This function frees all the resources associated with the respective JSON Generator.

### Usage

```
STATUS NU_JSON_Generator_Destroy(JSON_GEN_HANDLE *handle);
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) whose resource should be relinquished.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully, all resources were freed.
- **< 0**  
Error codes.

### Examples

```
JSON_GEN_HANDLE    *handle;
STATUS              status;
#define BUFFER_SIZE 512

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* A JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Construct and send a JSON Document. */

    /* Free resources associated with a JSON Generator. */
    NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

### Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Create](#)

## NU\_JSON\_Generator\_Start-Token

Appends a start of a token type to the JSON Generator.

### Usage

```
STATUS NU_JSON_Generator_Start-Token(JSON_GEN_HANDLE *handle
                                     CHAR             *name,
                                     UINT8            token_type)
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **name**  
Token's name. If NU\_NULL is passed in then no name is tied to the token. You should only specify this if the token itself is part of an Object.
- **token\_type**  
The type of token that is being added to the JSON string, which should be either:  
[JSON\\_TOKEN\\_TYPE\\_OBJECT](#)  
[JSON\\_TOKEN\\_TYPE\\_ARRAY](#)

### Return Values

- **NU\_SUCCESS**  
Function completed successfully, the token start was added.
- **< 0**  
Error codes.

### Examples

```
/* Sample JSON String being built */

JSON_GEN_HANDLE    *handle;
STATUS             status;
#define BUFFER_SIZE 512

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* A JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document. */
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                JSON_TOKEN_TYPE_OBJECT);
}
```

```
/* Start a Person Object. */
NU_JSON_Generator_Start-Token(&handle, "person",
                             JSON_TOKEN_TYPE_OBJECT);
/* Complete constructing the JSON String. */

/* Terminate JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Free resources associated with a JSON Generator. */

NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_End-Token](#)

## NU\_JSON\_Generator\_End\_Token

Appends the end of a token type to the JSON Generator.

### Usage

```
STATUS NU_JSON_Generator_End_Token(JSON_GEN_HANDLE *handle,  
                                   UINT8           token_type);
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **token\_type**  
The type of the token that this API will end, which should be either:  
  
`JSON_TOKEN_TYPE_OBJECT`  
`JSON_TOKEN_TYPE_ARRAY`

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the token was added.
- **< 0**  
Error codes.

### Examples

```
/* Sample JSON Document being built */  
  
JSON_GEN_HANDLE    *handle;  
STATUS             status;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator. */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
  
/* A JSON Generator was created. */  
if(status == NU_SUCCESS)  
{  
    /* Start a JSON Document. */  
    NU_JSON_Generator_Start_Token(&handle, NU_NULL,  
                                   JSON_TOKEN_TYPE_OBJECT);  
  
    /* Start a Person Object. */  
    NU_JSON_Generator_Start_Token(&handle, "person",  
                                   JSON_TOKEN_TYPE_OBJECT);  
  
    /* Finish constructing the JSON Document. */  
  
    /* End the Person Object. */  
    NU_JSON_Generator_End_Token(&handle, JSON_TOKEN_TYPE_OBJECT);  
}
```

```
    /* Terminate the JSON Document. */  
    NU_JSON_Generator_End_Token(&handle, JSON_TOKEN_TYPE_OBJECT);  
  
    /* Free the resources associated with a JSON Generator. */  
    NU_JSON_Generator_Destroy(&handle);  
}  
/* There was an error. */  
else  
{  
    /* Report error. */  
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Start\\_Token](#)

## NU\_JSON\_Generator\_Add\_Name

This function appends a name to the end of the JSON Document represented by this Generator.

### Usage

```
STATUS NU_JSON_Generator_Add_Name(JSON_GEN_HANDLE *handle,  
                                  CHAR              *name);
```

### Arguments

- handle  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- name  
The name part in a name/value pair.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the data was added to the buffer.
- < 0  
Error codes,
- NUF\_JSON\_BUFFER\_FULL  
The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to send this buffer of data, and then make additional calls to `NU_JSON_Generator_Add_Name` with the same "name" input, until this API returns `NU_SUCCESS`. The caller is not responsible for keeping up with the index into "name" at this point. The internal code will remember how much of "name" has been added to the string and add the remainder as subsequent calls are made.

### Examples

```
/* Sample JSON Document being built */  
  
JSON_GEN_HANDLE    *handle;  
STATUS              status;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator. */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
  
/* A JSON Generator was created. */  
if(status == NU_SUCCESS)  
{  
    /* Start a JSON Document*/  
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,  
                                  JSON_TOKEN_TYPE_OBJECT);
```



```
/* Start a Person Object. */
NU_JSON_Generator_Start-Token(&handle, "person",
                              JSON_TOKEN_TYPE_OBJECT);

/* Add a Person Object Properties.*/
status = NU_JSON_Generator_Add_Name(&handle, "first_name");

if (status == NU_SUCCESS)
{
    /* Finish constructing the JSON Document. */

    /* End the Person Object. */
    NU_JSON_Generator_End-Token(&handle,
                               JSON_TOKEN_TYPE_OBJECT);

    /* Terminate the JSON Document. */
    NU_JSON_Generator_End-Token(&handle,
                               JSON_TOKEN_TYPE_OBJECT);

    /* Free resources associated with a JSON Generator. */
    NU_JSON_Generator_Destroy(&handle);
}
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_String](#)

## NU\_JSON\_Generator\_Add\_String

This function appends a string to the end of the JSON Document. The function is capable of handling strings in chunks so that very large strings can be specified in multiple iterations.

### Usage

```
STATUS NU_JSON_Generator_Add_String(JSON_GEN_HANDLE *handle,  
                                   CHAR               *value,  
                                   UINT8              flags);
```

### Arguments

- handle  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with internal state of the JSON Generator.
- value  
Pointer to a string to be added.
- flags  
Used to specify whether the string data given to this API is complete or partial. The last chunk of data provided will have this flag cleared out to signify the end of the string. When the data is only a partial value, and more string data will be provided by subsequent calls to this API, this flag is:
  - JSON\_IS\_PARTIAL

### Return Values

- NU\_SUCCESS  
Function completed successfully, all data was added to the buffer.
- < 0  
Error codes.
- NUF\_JSON\_BUFFER\_FULL  
The buffer is full or the data added caused the buffer to become full. If that is the case, you will need to send this buffer of data, and then make additional calls to this API with the same parameters, until NU\_SUCCESS is returned.

### Examples

```
/ * Sample JSON String being built */  
  
JSON_GEN_HANDLE *handle;  
CHAR             *data;  
STATUS           status;  
UINT64           data_length;  
#define          BUFFER_SIZE 512  
CHAR buffer[BUFFER_SIZE]CHAR *long_str_part1 = "This is a really really  
                                             really";  
CHAR *long_str_part2 = "long string, used to demonstrate partial flag";
```

```

CHAR *str_exceed_buff = "This string will cause the JSON Generator to
                        exceed its buffer size."

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* A JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document. */
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                JSON_TOKEN_TYPE_OBJECT);

    /* Start an Object. */
    NU_JSON_Generator_Start-Token(&handle, "object",
                                JSON_TOKEN_TYPE_OBJECT);

    /* Add a Person Object Properties.*/

    /* Create a name/value pair. */
    NU_JSON_Generator_Add_Name(&handle, "string");
    NU_JSON_Generator_Add_String(&handle, "this is my string", 0);

    /******
    * Example using Partial Flag
    *******/
    /* Create a name/value pair demonstration using partial flag. */
    NU_JSON_Generator_Add_Name(&handle, "string");

    /* Let the Generator know that we need to add more to this string. */
    NU_JSON_Generator_Add_String(&handle, long_str_part1,
                                JSON_IS_PARTIAL);
    NU_JSON_Generator_Add_String(&handle, long_str_part2, 0);

    /******
    * Example using API returning value NUF_JSON_BUFFER_FULL
    * This means the buffer is full.
    *******/
    /* Create a name/value pair demonstration using partial flag. */
    NU_JSON_Generator_Add_Name(&handle, "str_grt_buff");

    /* Let the Generator know that we need to add more to this string. */
    status = NU_JSON_Generator_Add_String(&handle,
                                         str_exceed_buff , 0);

    /* If the JSON buffer reached its limit and wasn't able to add all of
    * str_grt_buff's value. */
    if(status == NUF_JSON_BUFFER_FULL)
    {
        /* Send data via networking interface. */

        /* Clear out the data generated by our JSON Generator. */
        NU_JSON_Generator_Clear_Buffer(&handle);

        status = NU_JSON_Generator_Add_String(&handle,
                                             str_exceed_buff, 0);
    }
}
/* End Object. */

```

```
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate JSON Document*/
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get pointer to the generated data, so we can send it.*/
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the generated data. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_Name](#)

## NU\_JSON\_Generator\_Add\_Boolean

Appends a Boolean to the end of a JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Add_Boolean(JSON_GEN_HANDLE *handle,
                                     BOOLEAN          value);
```

### Arguments

- **handle**  
 Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **value**  
 A Boolean to be added to the JSON Document.

### Return Values

- **NU\_SUCCESS**  
 The function completed successfully, the Boolean data was added to the buffer.
- **< 0**  
 Error codes.
- **NUF\_JSON\_BUFFER\_FULL**  
 The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to process the current JSON Generator buffer and clear it, and then make additional calls to this API until **NU\_SUCCESS** is returned.

### Examples

```
/* Sample JSON String being built */

JSON_GEN_HANDLE    *handle;
CHAR               *data;
STATUS              status;
#define BUFFER_SIZE 512
CHAR               buffer[BUFFER_SIZE];

/* Create a new JSON Generator */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* A JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document.*/
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                  JSON_TOKEN_TYPE_OBJECT);

    /* Start an Object. */
    NU_JSON_Generator_Start-Token(&handle, "object",
                                  JSON_TOKEN_TYPE_OBJECT);
```

```
/* Add a Person Object Properties.*/

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "string");
NU_JSON_Generator_Add_String(&handle, "this is my string", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "boolean");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* End the Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get the pointer to the generated data, so we can send it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */

NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_Float](#)

## NU\_JSON\_Generator\_Add\_Int

Appends a signed integer to the end of the JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Add_Int (JSON_GEN_HANDLE *handle,  
                                INT64             value);
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **value**  
The value to be added.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the data was added to the buffer.
- **< 0**  
Error codes.
- **NUF\_JSON\_BUFFER\_FULL**  
The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to process the current buffer and clear it, and then make additional calls to this API until **NU\_SUCCESS** is returned.

### Examples

```
/* Sample JSON Document being built */  
  
JSON_GEN_HANDLE    *handle;  
CHAR               *data;  
STATUS             status;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
/* JSON Generator was created. */  
if(status == NU_SUCCESS)  
{  
    /* Start a JSON Document. */  
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,  
                                JSON_TOKEN_TYPE_OBJECT);  
  
    /* Start an Object. */  
    NU_JSON_Generator_Start-Token(&handle, "object",  
                                JSON_TOKEN_TYPE_OBJECT);  
  
    /* Add a Person Object Properties.*/  
}
```

```
/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "string");
NU_JSON_Generator_Add_String(&handle, "this is my string", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "boolean");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "int");
NU_JSON_Generator_Add_Int(&handle, -1);

/* End the Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get a pointer to the generated data, so we can send it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send the data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_UInt](#)



## NU\_JSON\_Generator\_Add\_UInt

Appends an unsigned integer to the end of the JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Add_UInt (JSON_GEN_HANDLE *handle,  
                                   UINT64           value);
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **value**  
Value to be added to the JSON Document.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully, the data was added to the buffer.
- **< 0**  
Error codes.
- **NUF\_JSON\_BUFFER\_FULL**  
The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to process the current buffer and clear it, and then make additional calls to this API until **NU\_SUCCESS** is returned.

### Examples

```
/* Sample JSON String being built */  
  
JSON_GEN_HANDLE    *handle;  
CHAR               *data;  
STATUS             status;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
/* JSON Generator was created. */  
if(status == NU_SUCCESS)  
{  
    /* Start a JSON Document. */  
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,  
                                   JSON_TOKEN_TYPE_OBJECT);  
  
    /* Start an Object. */  
    NU_JSON_Generator_Start-Token(&handle, "object",  
                                   JSON_TOKEN_TYPE_OBJECT);  
  
    /* Add a Person Object Properties.*/  
}
```

```
/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "string");
NU_JSON_Generator_Add_String(&handle, "this is my string", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "boolean");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "int");
NU_JSON_Generator_Add_Int(&handle, -1);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "uint");
NU_JSON_Generator_Add_UInt(&handle, 1);

/* End the Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get a pointer to the generated data, so we can send it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_Int](#)

## NU\_JSON\_Generator\_Add\_Float

Appends a Float to the end of the JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Add_Float (JSON_GEN_HANDLE *handle,  
                                   CHAR *value);
```

### Arguments

- **handle**  
Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.
- **value**  
Pointer to the value to be added to the JSON Document.

### Return Values

- **NU\_SUCCESS**  
Function completed successfully, the data was added to the buffer.
- **< 0**  
Error codes.
- **NUF\_JSON\_BUFFER\_FULL**  
The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to process the current buffer and clear it, and then make additional calls to this API until **NU\_SUCCESS** is returned.

### Examples

```
/* Sample JSON String being built */  
  
JSON_GEN_HANDLE    *handle;  
CHAR               *data;  
STATUS             status;  
UINT64            data_length;  
#define BUFFER_SIZE 512  
  
/* Create a new JSON Generator */  
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);  
/* JSON Generator was created. */  
if(status == NU_SUCCESS)  
{  
    /* Start a JSON Document. */  
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,  
                                  JSON_TOKEN_TYPE_OBJECT);  
  
    /* Start an Object. */  
    NU_JSON_Generator_Start-Token(&handle, "object",  
                                  JSON_TOKEN_TYPE_OBJECT);  
  
    /* Add a Person Object Properties.*/
```

```
/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "string");
NU_JSON_Generator_Add_String(&handle, "this is my string", 0);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "boolean");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "int");
NU_JSON_Generator_Add_Int(&handle, -1);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "uint");
NU_JSON_Generator_Add_UInt(&handle, 1);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "float");
NU_JSON_Generator_Add_Float(&handle, "12.12");

/* End the Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get a pointer to the generated data, so we can send it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_Null](#)

## NU\_JSON\_Generator\_Add\_Null

Appends null to the end of the JSON Document.

### Usage

```
STATUS NU_JSON_Generator_Add_Null(JSON_GEN_HANDLE *handle);
```

### Arguments

- handle

Pointer to the [JSON\\_GEN\\_HANDLE](#) used to keep up with the internal state of the JSON Generator.

### Return Values

- NU\_SUCCESS

The function completed successfully, null was appended to the buffer.

- < 0

Error codes.

- NUF\_JSON\_BUFFER\_FULL

The buffer is full or the data added caused the buffer to become full. If the data added caused the buffer to become full you will need to process the current buffer and clear it, and then make additional calls to this API until NU\_SUCCESS is returned.

### Examples

```
/* Sample JSON String being built */

JSON_GEN_HANDLE    *handle;
CHAR               *data;
STATUS             status;
UINT64             data_length;
#define BUFFER_SIZE 512

/* Create a new JSON Generator */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);
/* JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document. */
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                   JSON_TOKEN_TYPE_OBJECT);

    /* Start an Object. */
    NU_JSON_Generator_Start-Token(&handle, "object",
                                   JSON_TOKEN_TYPE_OBJECT);

    /* Add a Person Object Properties.*/

    /* Create a name/value pair. */
    NU_JSON_Generator_Add_Name(&handle, "string");
    NU_JSON_Generator_Add_String(&handle, "this is my string", 0);
}
```

```
/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "boolean");
NU_JSON_Generator_Add_Boolean(&handle, NU_TRUE);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "int");
NU_JSON_Generator_Add_Int(&handle, -1);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "uint");
NU_JSON_Generator_Add_UInt(&handle, 1);

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "float");
NU_JSON_Generator_Add_Float(&handle, "12.12");

/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "nullable");
NU_JSON_Generator_Add_Null(&handle);

/* End the Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get a pointer to the generated data, so we can send it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Add\\_Float](#)

## NU\_JSON\_Generator\_Get\_Buffer

Returns a pointer to the JSON Generator's buffer that contains the current JSON Document, and that buffer's length, so the application can perform I/O operations on the buffer.

### Usage

```
STATUS NU_JSON_Generator_Get_Buffer(JSON_GEN_HANDLE *handle,
                                   CHAR               **buffer,
                                   UINT32             *buffer_length);
```

### Arguments

- **handle**  
 Pointer to the [JSON\\_GEN\\_HANDLE](#) whose buffer information is being requested.
- **buffer**  
 On successful return of this API, this buffer contains a pointer to an internal buffer of the JSON Generator which contains the JSON Document being generated. The returned data is null-terminated.
- **buffer\_length**  
 The total number of bytes being used in the buffer.

### Return Values

- **NU\_SUCCESS**  
 The function completed successfully.
- **< 0**  
 Error codes.

### Examples

```
/* Sample JSON String being sent */

JSON_GEN_HANDLE    *handle;
STATUS             status;
CHAR               *data;
UINT64             data_length;
#define BUFFER_SIZE 512

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);

/* A JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document. */
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                JSON_TOKEN_TYPE_OBJECT);

    /* Start a Person Object. */
```

```
    NU_JSON_Generator_Start-Token(&handle, "person",
                                   JSON_TOKEN_TYPE_OBJECT);

    /* Add a Person Object Properties.*/

    /* Create a name/value pair. */
    NU_JSON_Generator_Add_Name(&handle, "first_name");
    NU_JSON_Generator_Add_String(&handle, "Buffy", 0);

    /* Create a name/value pair. */
    NU_JSON_Generator_Add_Name(&handle, "last_name");
    NU_JSON_Generator_Add_String(&handle, "Summers", 0);

    /* End the Person Object. */
    NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

    /* Terminate the JSON Document. */
    NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

    /* Get a pointer to the generated data, so we can send it. */
    NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

    /* Send data via networking interface. */

    /* Free resources associated with a JSON Generator. */
    NU_JSON_Generator_Destroy(&handle);
}
/* There was an error */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Clear\\_Buffer](#)



## NU\_JSON\_Generator\_Clear\_Buffer

Clears out the contents of the JSON Generator's buffer, thus emptying the JSON Document. The buffer is not deallocated with this call. The purpose of this call is to continue building a JSON Document that is larger than the buffer after the previous buffer of data has been processed.

### Usage

```
STATUS NU_JSON_Generator_Clear_Buffer(JSON_GEN_HANDLE *handle);
```

### Arguments

- handle  
Pointer to the [JSON\\_GEN\\_HANDLE](#) whose resources should be relinquished.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the Generator's internal buffer was cleared.
- < 0  
Error codes.

### Examples

```
/* Sample JSON String being sent */

JSON_GEN_HANDLE    *handle;
CHAR               *data;
STATUS             status;
UINT64             data_length;
#define BUFFER_SIZE 512

/* Create a new JSON Generator. */
status = NU_JSON_Generator_Create(BUFFER_SIZE, &handle);
/* JSON Generator was created. */
if(status == NU_SUCCESS)
{
    /* Start a JSON Document. */
    NU_JSON_Generator_Start-Token(&handle, NU_NULL,
                                JSON_TOKEN_TYPE_OBJECT);

    /* Start a Person Object. */
    NU_JSON_Generator_Start-Token(&handle, "person",
                                JSON_TOKEN_TYPE_OBJECT);

    /* Add a Person Object Properties.*/

    /* Create a name/value pair. */
    NU_JSON_Generator_Add_Name(&handle, "first_name");
    NU_JSON_Generator_Add_String(&handle, "Buffy", 0);
}
```

```
/* Create a name/value pair. */
NU_JSON_Generator_Add_Name(&handle, "last_name");
NU_JSON_Generator_Add_String(&handle, "Summers", 0);

/* End the Person Object. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Terminate the JSON Document. */
NU_JSON_Generator_End-Token(&handle, JSON_TOKEN_TYPE_OBJECT);

/* Get pointer to the generated data, so we can sen it. */
NU_JSON_Generator_Get_Buffer(&handle, &data, &data_length);

/* Send data via networking interface. */

/* Clear out the data generated by our JSON Generator. */
NU_JSON_Generator_Clear_Buffer(&handle);

/* Add more data to generator. */

/* Free resources associated with a JSON Generator. */
NU_JSON_Generator_Destroy(&handle);
}
/* There was an error. */
else
{
    /* Report error. */
}
```

## Related Topics

[JSON Generator APIs](#)

[NU\\_JSON\\_Generator\\_Get\\_Buffer](#)

## JSON Parser Overview

JavaScript Object Notation (JSON) is a data interchange format which is frequently used in Web Applications, and on the Internet, to transfer structured data in a machine independent format. JSON is quite compact compared to XML and it can easily be used as a replacement of XML. The JSON format is easily readable for humans and is also relatively easy for machines to parse. In fact, it can directly be evaluated as JavaScript since JSON is a subset of JavaScript.

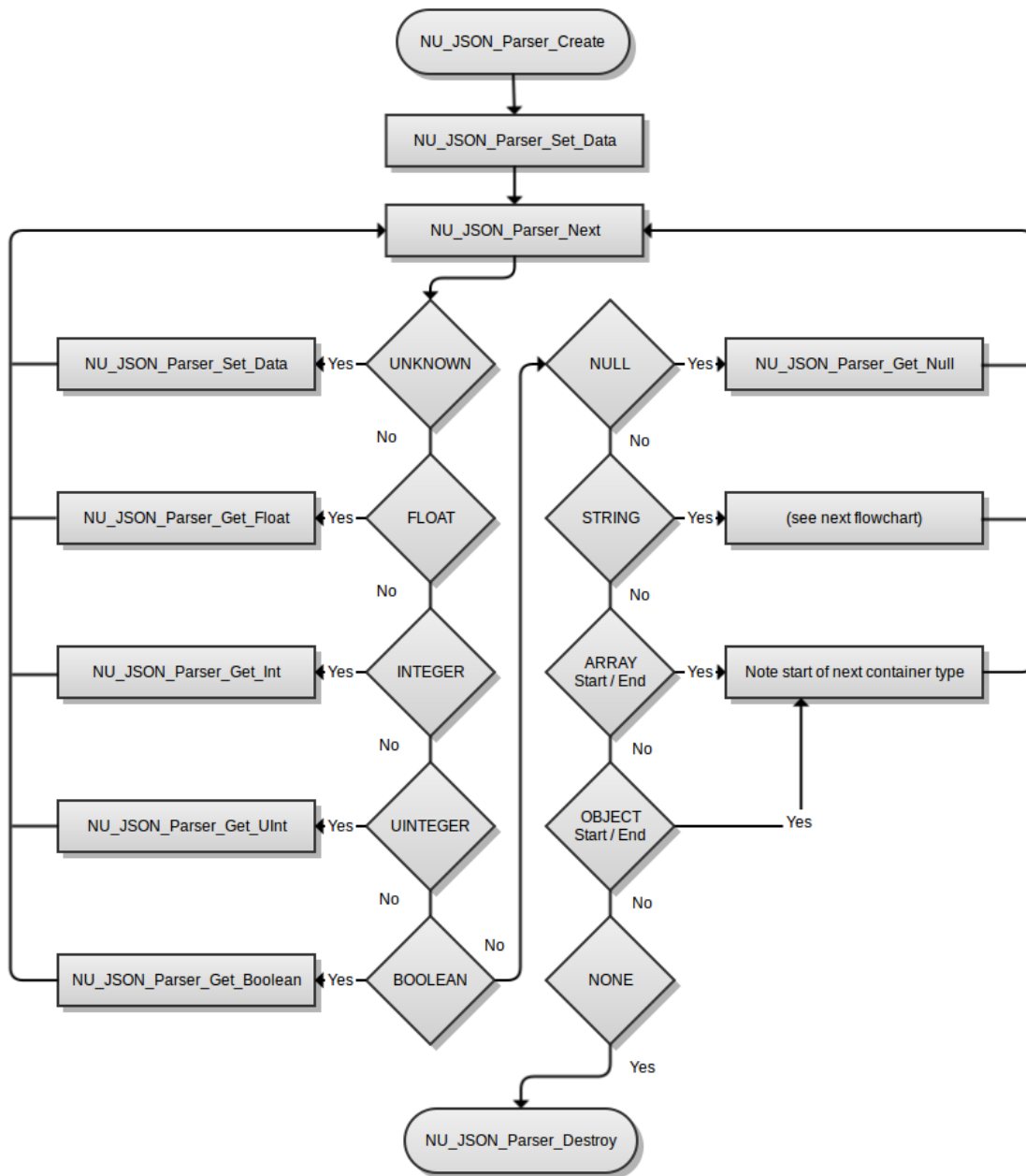
## Nucleus JSON Parser Implementation

Nucleus provides a JSON Parser which can be used to parse JSON data. It also validates the JSON data and ensures that it complies with the JSON grammar. If the data violates the grammar, the JSON Parser APIs return a parsing error. The JSON Parser also handles UTF-8 data correctly and reports an error if malformed UTF-8 data is encountered. The parser also handles Unicode escape characters correctly (which are generated using the "\u" escape sequence).

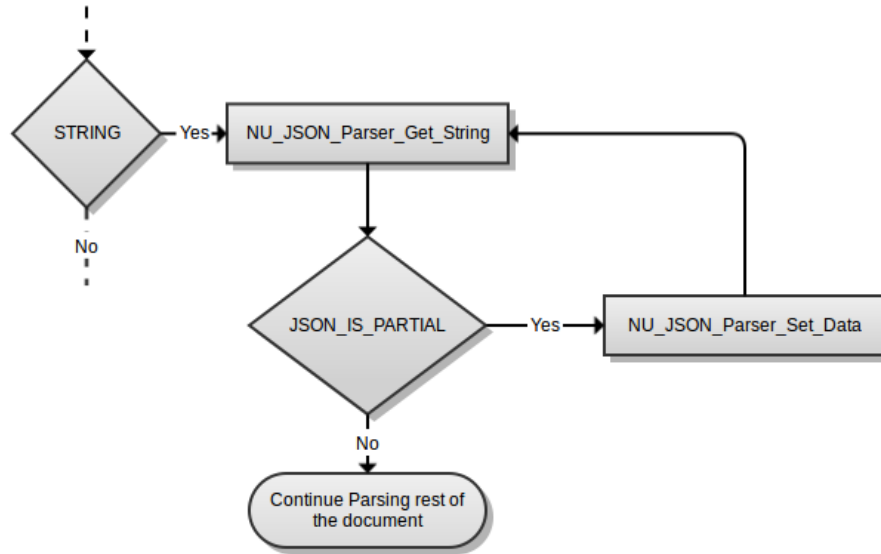
The Nucleus JSON Parser provides you with a set of API functions which can be used to traverse a JSON document. The basic procedure for parsing JSON is that you gradually feed JSON data to the parser, possibly as it comes through over the network. It is not mandatory to provide a complete JSON document to the parser at one time. You then call the [NU\\_JSON\\_Parser\\_Next](#) API to get the type for the next item of data and then call the corresponding "get" function for the reported data type to obtain its value. An "unknown" data type is returned if the JSON Parser does not have enough data in its buffer to determine the next data type. In that case you must feed more data into the JSON Parser and determine the next data type again.

This procedure is summarized in the flowchart in [Figure 17-1](#):

Figure 17-1. JSON Parser Core



The parsing of the "string" data type is special due to its size constraints. A string in JSON can be very large and may not fit in the buffer configured. The Nucleus JSON Parser therefore allows a string value to be obtained in multiple [NU\\_JSON\\_Get\\_String](#) API calls. Each time this API is called, it returns a flag specifying whether the string data returned is now complete or is partial. If the data is partial, then more data should be fed into the JSON Parser and then the same [NU\\_JSON\\_Get\\_String](#) API must be called again to obtain the next chunk of string data. This procedure is shown in the flowchart in [Figure 17-2](#).

**Figure 17-2. Parsing a String**

## Understanding the Stack Levels

The JSON Parser's [NU\\_JSON\\_Parser\\_Next](#) API function returns the current "nesting depth" or "stack level" of the next data type. This nesting depth is a numeric value which shows how deeply nested the next value is. Nesting depth can be understood using the [Figure 17-3](#).

**Figure 17-3. Stack Levels**

```
Level 0  _____ {
Level 1  _____ "person":{
Level 2  _____   "first_name":"Bob",
Level 2  _____   "last_name":"Smith",
Level 2  _____   "dog":{
Level 3  _____     "breed":"Boxer",
Level 3  _____     "age":5.6
Level 2  _____   },
Level 2  _____   "TV Shows":[
Level 3  _____     "Buffy",
Level 3  _____     "Farscape"
Level 2  _____   ],
Level 2  _____   "Cars":[
Level 3  _____     {
Level 4  _____       "4Runner":{
Level 5  _____         "color":"teal",
Level 5  _____         "cylinders":6,
Level 4  _____       }
Level 3  _____     },
Level 3  _____     {
Level 4  _____       "Tahoe":{
Level 5  _____         "color":"green",
Level 5  _____         "cylinders":8,
Level 4  _____       }
Level 3  _____     }
Level 2  _____   ]
Level 1  _____ }
Level 0  _____ }
```

## JSON Parser Data Structures and Constants

In order to understand the JSON Parser interface, you need to be aware of the following data structures and constants.

The JSON Parser constants are defined in [Table 17-1](#):

**Table 17-1. JSON Parser Constants**

Name	Description
JSON_TYPE_BOOLEAN	Data type.
JSON_TYPE_STRING	Data type.
JSON_TYPE_INTEGER	Data type.
JSON_TYPE_FLOAT	Data type.
JSON_TYPE_UINTEGER	Data type.

**Table 17-1. JSON Parser Constants (cont.)**

Name	Description
JSON_TYPE_NULL	Data type.
JSON_TYPE_OBJECT_START	Token type “object” start.
JSON_TYPE_OBJECT_END	Token type “object” end.
JSON_TYPE_BOOLEAN	Data type.
JSON_TYPE_ARRAY_START	Token type “array” start.
JSON_TYPE_ARRAY_END	Token type “array” end.
JSON_TYPE_UNKNOWN	Data type unknown. The parser does not currently have enough JSON data to determine the next type.
JSON_TYPE_NONE	The parser reached the end of the JSON document and has no more data to process.

## JSON\_PARSER\_HANDLE

The member of the JSON Parser structure is defined in [Table 17-2](#):

**Table 17-2. JSON Parser Structure**

Member	Description
JSON_PARSER_HANDLE	Public access to JSON Parser’s context via opaque pointer.

## JSON\_STRING

The following is the definition of the JSON\_STRING data structure:

```
typedef struct
{
    CHAR    *str;
    INT     length;
}JSON_STRING;
```

The members of the JSON\_STRING data structure are defined in [Table 17-3](#):

**Table 17-3. JSON Parser String Structure**

Member	Description
str	String data, null-terminated.
length	String length, also used to indicate the total buffer length to where “str” is pointing.

## JSON Parser APIs

This section describes the Nucleus JSON Parser APIs. In order to make use of these APIs, you need to include the following header file in your application as shown below:

```
#include "networking/nu_json.h"
```

- [NU\\_JSON\\_Parser\\_Create](#)
- [NU\\_JSON\\_Parser\\_Destroy](#)
- [NU\\_JSON\\_Parser\\_Set\\_Data](#)
- [NU\\_JSON\\_Parser\\_Reset](#)
- [NU\\_JSON\\_Parser\\_Next](#)
- [NU\\_JSON\\_Parser\\_Get\\_Info](#)
- [NU\\_JSON\\_Get\\_Boolean](#)
- [NU\\_JSON\\_Get\\_Int](#)
- [NU\\_JSON\\_Get\\_UInt](#)
- [NU\\_JSON\\_Get\\_Null](#)
- [NU\\_JSON\\_Get\\_String](#)
- [NU\\_JSON\\_Get\\_Float](#)



## NU\_JSON\_Parser\_Create

Creates a new JSON Parser context/handle.

### Usage

```
STATUS NU_JSON_Parser_Create(INT          buffer_size,  
                             JSON_PARSER_HANDLE *handle);
```

### Arguments

- `buffer_size`  
Size of the internal buffer used by the JSON Parser for holding JSON data. This has a minimum limit of 6 bytes and can be much larger.
- `handle`  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used with calling the JSON Parser API.

### Return Values

- `NU_SUCCESS`  
The function completed successfully, the JSON Parser was created.
- `< 0`  
Errors.

### Examples

```
JSON_PARSER_HANDLE json_parser_handle;  
  
/* Create our Parser. */  
status = NU_JSON_Parser_Create(100, &json_parser_handle);
```

### Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Destroy](#)

## NU\_JSON\_Parser\_Destroy

Frees the resources associated with the JSON Parser.

### Usage

```
STATUS NU_JSON_Parser_Destroy(JSON_PARSER_HANDLE *handle);
```

### Arguments

- handle  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used to keep up with the internal state of the JSON Parser.

### Return Values

- NU\_SUCCESS  
Function completed successfully, the resources were freed.
- < 0  
Errors

### Examples

```
CHAR                *json_string = NU_NULL;
JSON_PARSER_HANDLE json_parser_handle;
INT                 json_string_len;

/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &json_parser_handle);
do{
    /* json_string gets filled with data via
     * network interface, reading from a file,
     * and json_string_len gets filled with the length of the data.
     */

    NU_JSON_Parser_Set_Data(&json_parser_handle, json_string,
                           &json_string_len);

    /* Call remaining API's the handle parsing the string. */
} while(json_string != NU_NULL)

NU_JSON_Parser_Destroy(&json_parser_handle);
```

### Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Create](#)

## NU\_JSON\_Parser\_Set\_Data

Sets the JSON String to be parsed. If there is previously any unprocessed data in the JSON context, this new data is appended to the end of the previously specified data. This function copies the JSON data into the parser's internal buffer and returns the number of bytes of JSON data which were actually copied.

### Usage

```
STATUS NU_JSON_Parser_Set_Data(JSON_PARSER_HANDLE *handle,  
                               CHAR                *json_data,  
                               INT                 *json_data_length);
```

### Arguments

- **handle**  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used to keep up with the internal state of the JSON Parser.
- **json\_data**  
A JSON string to be parsed. The string must be null-terminated and it can be:
  - A complete JSON document
  - An incomplete chunk of JSON data from a document.
- **json\_data\_length**
  - On input you must specify the length of the data contained in the "json\_data" parameter.
  - On successful return, this contains the amount of data which was actually added to the Parser's internal buffer. If this is not equal to the total length of "json\_data", you must add the remaining data again after processing the data currently held in the Parser's internal buffer.

### Description

See the examples for “Complete JSON string” and “Incomplete JSON string” for an understanding of the data parser.

```
/* Example of a complete JSON string */  
{  "person": {  
    "first_name": "Bob",  
    "last_name": "Smith"  
  }  
}  
  
/* Example of an incomplete JSON String */  
{  "person": {  "first_name": "Bob",
```

### Examples

```
/* Usage example */
```

```
CHAR          *json_string = NU_NULL;
JSON_PARSER_HANDLE json_parser_handle;
INT           data_length;

/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &json_parser_handle);

do {
    /* json_string gets filled with data via
     * network interface or reading data from a file, etc.
     */
    data_length = strlen(json_string);
    NU_JSON_Parser_Set_Data(&json_parser_handle, json_string,
                           &data_length);

    if (data_length != strlen(json_string))
    {
        /* Not all bytes were copied. Save the last few remaining
         * bytes of "json_string" for processing in the next iteration.
         */
    }

    /* Call remaining API's the handle parsing the string. */
} while(json_string != NU_NULL)
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Reset](#)

## NU\_JSON\_Parser\_Reset

Resets the internal state of the JSON Parser so it is ready to start a new JSON document. This function also flushes all buffered data from the Parser so the [NU\\_JSON\\_Parser\\_Set\\_Data](#) API must be called again to feed new data into the Parser.

### Usage

```
STATUS NU_JSON_Parser_Reset (JSON_PARSER_HANDLE *handle);
```

### Arguments

- `handle`  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs.

### Return Values

- `NU_SUCCESS`  
The function completed successfully, the Parser's internal state was reset.
- `< 0`  
Errors.

### Examples

```
JSON_PARSER_HANDLE json_parser_handle;  
CHAR                *json_string = NU_NULL;  
  
/* Create our Parser. */  
status = NU_JSON_Parser_Create(100, &json_parser_handle);  
  
/* Parse the first JSON document. */  
...  
  
/* Reset the Parser without destroying the context. */  
status = NU_JSON_Parser_Reset(&json_parser_handle);  
  
/* Parse the second JSON document. */  
...  
/* Destroy the Parser context now. */  
status = NU_JSON_Parser_Destroy(&json_parser_handle);
```

### Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Set\\_Data](#)

## NU\_JSON\_Parser\_Next

Determines the next type found in the JSON data so you know which API to call to get the data.

### Note



If "JSON\_TYPE\_UNKNOWN" is returned then the arser does not currently have enough JSON data to determine the next type. This means the application will need to call this API again after feeding the parser more data using the [NU\\_JSON\\_Parser\\_Set\\_Data](#) API.

### Usage

```
STATUS NU_JSON_Parser_Next (JSON_PARSER_HANDLE *handle,  
                           UINT8                *type,  
                           UINT8                *stack_level,  
                           CHAR                 *name);
```

### Arguments

- **handle**  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the parser's APIs.
- **type**  
On return, type contains the next type. Valid types are described in [Table 17-3](#):
- **stack\_level**  
On return, stack\_level contains a numeric value specifying the current nesting depth of the type being reported. The top most nesting depth is 0, the next being 1 and so on.
- **name**  
You must pass a buffer of size (JSON\_MAX\_NAME\_LENGTH + 1) for this parameter. On return, if the next type has an associated name (for example, if it is a key value pair within a JSON Object), the name of that type is returned in this buffer. The name is null-terminated. A blank string is returned if the name is not applicable.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the next type was determined.
- **NUF\_JSON\_PARSING\_ERROR**  
Invalid data was encountered which does not comply with the JSON grammar.
- **NUF\_INVALID\_UTF8\_NAME**  
Invalid UTF-8 data found in the "name".
- **< 0**  
Errors.

## Description

The nesting depth of the next type is also returned by this API, which specifies how deeply the current data is nested. The next data may also have a "name" associated with it, if it is a key/value pair inside a JSON Object. This function returns a type of "JSON\_TYPE\_NONE" when it reaches the end of the JSON document and has no more data to process.

## Examples

```
STATUS          status;
JSON_PARSER_HANDLE handle;
CHAR            *json_string = NU_NULL;
UINT8           next_type;
UINT8           stack_level;
CHAR            name[JSON_MAX_NAME_LENGTH + 1];
INT             json_string_len;

/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &handle);

/* json_string gets filled with some JSON data
 * and json_string_len gets filled with its length.
 */
...

/* Feed some data into the Parser. */
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);
do {
    /* Determine what our next API call should be. */
    status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,
                                   name);
    if (next_type == JSON_TYPE_UNKNOWN)
    {
        /* json_string gets filled with more JSON data
         * and json_string_len gets filled with its length.
         * Any data left from previous call is also added.
         */
        ...
        /* Feed more data into the Parser. */
        status = NU_JSON_Parser_Set_Data(&handle, json_string,
                                         &json_string_len);
    }
    /* Process the other data types. */
    ...
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));
NU_JSON_Parser_Destroy(&handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Get\\_Info](#)

## NU\_JSON\_Parser\_Get\_Info

Gets information about the JSON parser and its internal state. The current and next indices returned by this function might not point exactly to the correct location and are approximations due to the limitation of how the JSON parser maintains the state internally.

### Usage

```
STATUS NU_JSON_Parser_Get_Info(JSON_PARSER_HANDLE *handle,  
                               BOOLEAN             compress,  
                               INT                 *current_type_index,  
                               INT                 *next_type_index,  
                               INT                 *buffer_left);
```

### Arguments

- **handle**  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs.
- **compress**  
Specifies whether "buffer\_left" represents compressed or uncompressed data size. Compressed data size would strip out all the white-space.
- **current\_type\_index**  
On return, this contains the position in the buffer of the current type. This is -1 if unknown. This is an optional parameter and can be NU\_NULL.
- **next\_type\_index**  
On return, this contains the position in the buffer of the next type. This is -1 if unknown. This is an optional parameter and can be NU\_NULL.
- **buffer\_left**  
On return, this contains the number of unprocessed bytes in the Parser's buffer. This is an optional parameter and can be NU\_NULL.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully and returned the Parser internal state data.
- **< 0**  
Errors

### Examples

```
JSON_PARSER_HANDLE handle;  
CHAR                *json_string = NU_NULL;  
INT                 current_type_index;  
INT                 next_type_index;  
INT                 buffer_left;
```



```
/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &jjson_parser_handle);
do {
    /* Parse some JSON data. */
    ...
    /* Check to see how much unprocessed data exists
     * in the parser. */
    status = NU_JSON_Parser_Get_Info(&handle, NU_FALSE,
                                     &current_type_index,
                                     &next_type_index,
                                     &buffer_left);
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));

NU_JSON_Parser_Destroy(&jjson_parser_handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Parser\\_Next](#)

## NU\_JSON\_Get\_Boolean

Gets the next Boolean value encountered by the JSON Parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_BOOLEAN" data type.

### Usage

```
STATUS NU_JSON_Parser_Get_Boolean(JSON_PARSER_HANDLE *handle,  
                                BOOLEAN *value);
```

### Arguments

- **handle**  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs.
- **value**  
On return, this contains the Boolean value, which is either NU\_TRUE or NU\_FALSE.

### Return Values

- **NU\_SUCCESS**  
The function completed successfully, the Boolean value encountered was returned.
- **NUF\_JSON\_UNEXPECTED\_TYPE**  
The JSON Parser did not expect to find a value of this type.
- **< 0**  
Errors.

### Examples

```
STATUS          status;  
JSON_PARSER_HANDLE handle;  
CHAR            *json_string = NU_NULL;  
UINT8           next_type;  
UINT8           stack_level;  
CHAR            name[JSON_MAX_NAME_LENGTH + 1];  
BOOLEAN         value;  
INT             json_string_len;  
  
/* Create our Parser. */  
status = NU_JSON_Parser_Create(100, &handle);  
  
/* json_string gets filled with some JSON data  
 * and json_string_len is set to its length. */  
...  
/* Feed some data into the Parser. */  
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);  
  
do {  
    /* Determine what our next API call should be. */  
    status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,  
                                name);
```

```
if (next_type == JSON_TYPE_BOOLEAN)
{
    /* Obtain the value. */
    status = NU_JSON_Parser_Get_Boolean(&handle, &value);
}
/* Process the other data types. */
...
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));

NU_JSON_Parser_Destroy(&handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Get\\_Int](#)

## NU\_JSON\_Get\_Int

Gets the next signed integer value encountered by the JSON parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_INTEGER" data type.

### Usage

```
STATUS NU_JSON_Parser_Get_Int (JSON_PARSER_HANDLE *handle,  
                              INT64               *value);
```

### Arguments

- handle  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs.
- value  
On return, value contains the signed integer value.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the next signed integer value was returned.
- NUF\_JSON\_UNEXPECTED\_TYPE  
The JSON parser did not expect to find a value of this type.
- < 0  
Errors.

### Examples

```
STATUS          status;  
JSON_PARSER_HANDLE handle;  
CHAR            *json_string = NU_NULL;  
UINT8           next_type;  
UINT8           stack_level;  
CHAR            name[JSON_MAX_NAME_LENGTH + 1];  
INT64           value;  
INT             json_string_len;  
  
/* Create parser. */  
status = NU_JSON_Parser_Create(100, &handle);  
  
/* json_string gets filled with some JSON data  
 * and json_string_len is set to its length.  
 */  
...  
/* Feed some data into the Parser. */  
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);  
do {  
    /* Determine what our next API call should be. */  
    status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,  
                                   name);
```

```
if (next_type == JSON_TYPE_INTEGER)
{
    /* Obtain the value. */
    status = NU_JSON_Parser_Get_Int(&handle, &value);
}
/* Process the other data types. */
...
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));

NU_JSON_Parser_Destroy(&handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Get\\_UInt](#)

## NU\_JSON\_Get\_UInt

Gets the next unsigned integer value encountered by the JSON Parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_INTEGER" data type.

### Usage

```
STATUS NU_JSON_Parser_Get_UInt (JSON_PARSER_HANDLE *handle,  
                               UINT64               *value);
```

### Arguments

- handle  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs
- value  
On return value contains the unsigned integer value.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the next unsigned integer value was returned.
- NUF\_JSON\_UNEXPECTED\_TYPE  
The JSON Parser did not expect to find a value of this type.
- < 0  
Errors.

### Examples

```
STATUS          status;  
JSON_PARSER_HANDLE handle;  
CHAR            *json_string = NU_NULL;  
UINT8           next_type;  
UINT8           stack_level;  
CHAR            name[JSON_MAX_NAME_LENGTH + 1];  
UINT64          value;  
INT             json_string_len;  
  
/* Create our Parser. */  
status = NU_JSON_Parser_Create(100, &handle);  
  
/* json_string gets filled with some JSON data  
 * and json_string_len is set to its length.  
 */  
...  
  
/* Feed some data into the Parser. */  
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);  
do {
```

```
/* Determine what our next API call should be. */
status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,
                               name);
if (next_type == JSON_TYPE_UINTEGER)
{
    /* Obtain the value. */
    status = NU_JSON_Parser_Get_UInt(&handle, &value);
}
/* Process the other data types. */
...
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));

NU_JSON_Parser_Destroy(&handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Get\\_Int](#)

## NU\_JSON\_Get\_Null

Gets the next null value encountered by the JSON Parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_NULL" data type.

### Usage

```
STATUS NU_JSON_Parser_Get_Null (JSON_PARSER_HANDLE *handle);
```

### Arguments

- handle  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the parser's APIs.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the next null value encountered was returned.
- NUF\_JSON\_UNEXPECTED\_TYPE  
The JSON Parser did not expect to find this type.
- < 0  
Errors.

### Examples

```
STATUS          status;
JSON_PARSER_HANDLE handle;
CHAR            *json_string = NU_NULL;
UINT8           next_type;
UINT8           stack_level;
CHAR            name[JSON_MAX_NAME_LENGTH + 1];
INT             json_string_len;

/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &handle);

/* json_string gets filled with some JSON data
 * and json_string_len is set to its length.
 */
...
/* Feed some data into the Parser. */
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);

do {
    /* Determine what our next API call should be. */
    status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,
                                   name);

    if (next_type == JSON_TYPE_NULL)
    {
        /* Obtain the value. */
        status = NU_JSON_Parser_Get_Null(&handle);
    }
}
```



```
    /* Process the other data types. */  
    ...  
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));  
  
NU_JSON_Parser_Destroy(&handle);
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Get\\_String](#)

## NU\_JSON\_Get\_String

Gets the next string value encountered by the JSON parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_STRING" data type. A string in a JSON document can be quite long so this API can be called multiple times to obtain the value of a single string in multiple chunks of data.

### Usage

```
STATUS NU_JSON_Parser_Get_String(JSON_PARSER_HANDLE *handle,  
                                JSON_STRING         *value,  
                                UINT8               *flags);
```

### Arguments

- handle

A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the Parser's APIs.

- value

On input, the "str" member of the [JSON\\_STRING](#) structure must point to a caller-provided buffer and the "length" member of this structure must contain the length of this buffer in bytes. On successful return, the string data is returned in the "str" buffer and its length is reported in the "length" member. The string data returned by this function is null-terminated.

- flags
  - JSON\_IS\_PARTIAL

This is used to indicate that the parser has completed processing the current data, but the string returned by this API is incomplete. This means that this API should be called immediately after [NU\\_JSON\\_Parser\\_Set\\_Data](#) is called again.

If this flag is not set, the string returned is considered complete and you can continue parsing the rest of the JSON document.

### Return Values

- NU\_SUCCESS  
The function completed successfully, the next string value was returned.
- NUF\_JSON\_UNEXPECTED\_TYPE  
The JSON parser did not expect to find a value of this type.
- < 0  
Errors.

### Examples

```
STATUS      status;  
UINT8      flags;
```

```
JSON_STRING str_value;
CHAR        buffer[20];
INT         json_string_len;
...

/* Create a Parser context. */
...

/* Read data into "json_string" and give it to the Parser,
 * also update "json_string_len" to contain its length.
 */

NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);

/* Determine the next data type. */
status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level, name);

if (next_type == JSON_TYPE_STRING)
{
    do {
        flags = 0;
        if (flags & JSON_IS_PARTIAL)
        {
            /* Read more data into "json_string" and give it to the Parser,
             * also update "json_string_len" to contain its length.
             */
            NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);
        }
        str_value.str = buffer;
        str_value.length = sizeof(buffer);
        status = NU_JSON_Parser_Get_String(&handle, &str_value, &flags);
        if (status == NU_SUCCESS)
        {
            /* Process string data. */
        }
    } while ((status == NU_SUCCESS) && (flags & JSON_IS_PARTIAL));

    /* Process other data types. */
    ...
}
```

## Related Topics

[JSON Parser APIs](#)

[NU\\_JSON\\_Get\\_Null](#)

## NU\_JSON\_Get\_Float

Gets the next float value encountered by the JSON parser. This API must only be called if the [NU\\_JSON\\_Parser\\_Next](#) API reports a "JSON\_TYPE\_FLOAT" data type. To avoid rounding errors, the JSON parser returns floating point values in null-terminated string format. A maximum float value of length JSON\_MAX\_FLOAT\_LENGTH can be handled by the JSON parser.

### Note



The JSON parser provides utility functions for converting values from "double" to "string" formats and vice-versa. These APIs are "JSON\_String\_To\_Double" and "JSON\_Double\_To\_String" which can be optionally enabled in the build. See the JSON parser's configuration options for more information.

---

## Usage

```
STATUS NU_JSON_Parser_Get_Float(JSON_PARSER_HANDLE *handle,  
                                JSON_STRING         *value);
```

## Arguments

- **handle**  
A pointer to the [JSON\\_PARSER\\_HANDLE](#), used when calling the parser's APIs.
- **value**  
On input, the "str" member of the [JSON\\_STRING](#) structure must point to a caller-provided buffer and the "length" member of this structure must contain the length of this buffer in bytes. On successful return, the float data is returned in the "str" buffer and its length is reported in the "length" member. The numeric value in string format returned by this function is null-terminated.

## Return Values

- **NU\_SUCCESS**  
The function completed successfully, the next float value was returned.
- **NUF\_JSON\_UNEXPECTED\_TYPE**  
The JSON parser did not expect to find a value of this type.
- **< 0**  
Errors.

## Examples

```
STATUS          status;
JSON_PARSER_HANDLE handle;
CHAR            *json_string = NU_NULL;
UINT8           next_type;
UINT8           stack_level;
CHAR            name[JSON_MAX_NAME_LENGTH + 1];
JSON_STRING     value;
CHAR            buffer[JSON_MAX_FLOAT_LENGTH + 1];
INT             json_string_len;

/* Create our Parser. */
status = NU_JSON_Parser_Create(100, &handle);

/* json_string gets filled with some JSON data
 * and json_string_len is set to its length.
 */
...
/* Feed some data into the Parser. */
status = NU_JSON_Parser_Set_Data(&handle, json_string, &json_string_len);
do {
    /* Determine what our next API call should be. */
    status = NU_JSON_Get_Next_Type(&handle, &next_type, &stack_level,
                                   name);
    if (next_type == JSON_TYPE_FLOAT)
    {
        value.str = buffer;
        value.length = sizeof(buffer);

        /* Obtain the value. */
        status = NU_JSON_Parser_Get_Float(&handle, &value);
    }
    /* Process the other data types. */
    ...
} while ((STATUS == NU_SUCCESS) && (next_type != JSON_TYPE_NONE));

NU_JSON_Parser_Destroy(&json_parser_handle);
```

## Related Topics

[JSON Parser APIs](#)[NU\\_JSON\\_Get\\_Int](#)



# Chapter 18


## Wireless Protected Access (WPA)

---

This chapter describes the usage and configuration of the Nucleus WPA Supplicant. This product is used with other Nucleus products to implement enhanced security features defined by the IEEE 802.11i and 802.1X standards.

Nucleus WPA Supplicant is based on the IEEE 802.11i and 802.1X standards. For more information on these standards, visit <http://www.ieee.org/>.

---

 **Warning** In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus reference guides. There is no guarantee of future support or compatibility for any interface that is not documented.

---

## WLAN Overview

The IEEE 802.11 standard and its set of amendments define the Wireless LAN (WLAN) communication protocol. This family of standards addresses various aspects of WLAN communication. Security is a very important domain for WLAN networks. A WLAN network has no physical connections. Users at a physically distant location can access a WLAN network and easily gain unauthorized access to the network.

The security defined by the IEEE 802.11 standard has had several revisions in the past. The initial security mechanism defined by the IEEE 802.11 standard had flaws and did not provide strong security. An “Enhanced Security” amendment titled IEEE 802.11i was initiated to address the weaknesses of the base standard. The Wi-Fi Alliance took a subset of the 802.11i standard that could be implemented using existing hardware at that time. This was named the “Wireless Protected Access” (WPA). When the IEEE 802.11i standard was approved, it was named “Wireless Protected Access 2” (WPA2).

## Nucleus WPA Supplicant

The Nucleus WPA Supplicant is an implementation of the 802.11 security methods for a client station defined by IEEE 802.11i. This product supports both WPA and WPA2 security mechanisms. It allows the client station to authenticate and associate with a WLAN server station (that is, an Access Point).

## Key Features and Supported Standards

The following are the 802.11i features supported by the Nucleus WPA Supplicant:

- WPA-PSK (“WPA-Personal” mode).
- WPA using EAP methods (“WPA-Enterprise” mode).
- WPA and WPA2/802.11i/RSN.
- Key management for CCMP, TKIP and WEP.

The following are the EAP methods supported by the Nucleus WPA Supplicant:

- EAP-TLS.
- EAP-PEAP/MSCHAPv2.
- EAP-PEAP/TLS.
- EAP-PEAP/MD5-Challenge.

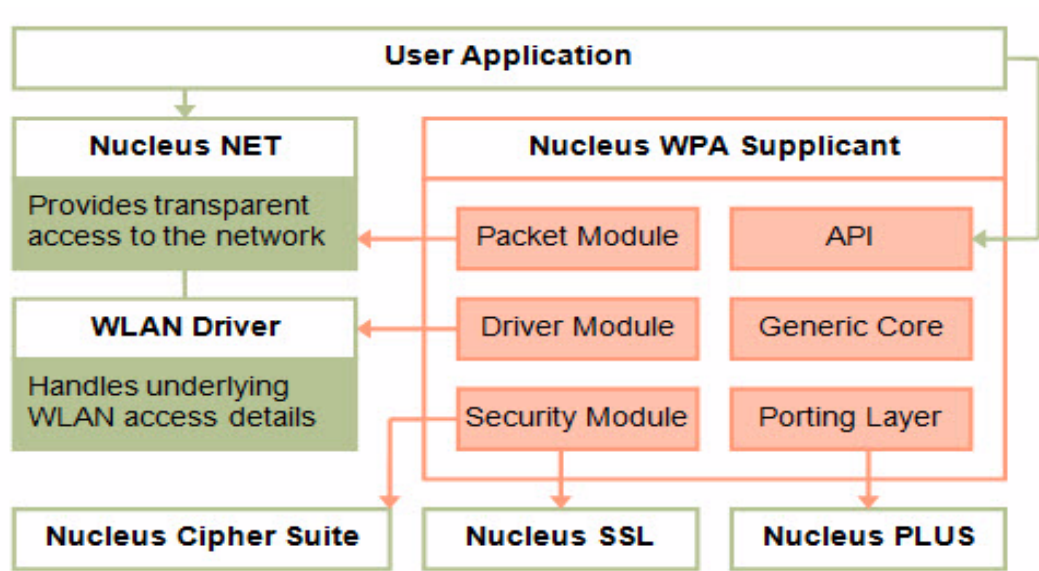
Additional features of the Nucleus WPA Supplicant are as follows:

- Hot-swapping of the WLAN interface (the WLAN card is on a bus that supports hot-swapping, such as USB or SDIO).
- Re-configuration at run-time.

## WPA Supplicant Architecture

[Figure 18-1](#) shows how the Nucleus WPA Supplicant interfaces with various Nucleus products. The Nucleus WPA Supplicant fully integrates with Nucleus NET and the Wireless LAN driver. It runs transparently with Nucleus NET, so existing user applications that run over Ethernet can now also run over Wireless LAN. Similarly, all Nucleus Networking Middleware products are compatible with WLAN and the Nucleus WPA Supplicant.



**Figure 18-1. WPA Supplicant with Nucleus Products**

Nucleus WPA Supplicant also integrates with OpenSSL for various security services required by IEEE 802.11i.

The Nucleus WPA Supplicant has an OS porting layer that allows it to be ported to various operating systems. The Nucleus product ships with a porting layer that uses Nucleus PLUS for various operating system services. This architecture allows a large portion of the WPA Supplicant's generic core to remain unmodified.

The Nucleus WPA Supplicant has a very small and simple API. Most of its configuration parameters are specified using its configuration file. The configuration file has a well-known format used by the Open Source "wpa\_supplicant" project. Although not shown in the diagram, Nucleus WPA Supplicant can also use the Nucleus FILE product to read its configuration file from secondary storage (such as an SD memory card).

## Open Source "wpa\_supplicant" Project

The Mentor Graphics' Nucleus WPA Supplicant implementation is based on the Open Source "wpa\_supplicant" project managed by Jouni Malinen [j@w1.fi](mailto:j@w1.fi). Most of the generic source components of the Nucleus WPA Supplicant are unmodified versions of the Open Source code. Therefore, almost all documentation and resources on the Internet for the "wpa\_supplicant" project are also applicable to Nucleus WPA Supplicant (provided that the correct version is referenced). See the following link for more information about the Open Source "wpa\_supplicant" project:

[http://hostap.epitest.fi/wpa\\_supplicant/](http://hostap.epitest.fi/wpa_supplicant/)

**Note**



The release notes for the Nucleus WPA Supplicant product specify the “wpa\_supplicant” version from which the Nucleus product was created. The features of the Open Source project for this version are comparable to the Nucleus product.

---

## Nucleus WPA Supplicant API Functions

This section describes the APIs provided by the Nucleus WPA Supplicant. To make use of these APIs, include the file `<wpa_supplicant/wpa_supplicant_api.h>` in your application.

If you want to use the Control Interface of wpa\_supplicant, include the following header files in your application (in this order):

```
wpa_supplicant/src/utils/includes.h  
wpa_supplicant/src/utils/common.h  
wpa_supplicant/src/utils/os.h  
wpa_supplicant/src/common/defs.h  
wpa_supplicant/src/common/wpa_ctrl.h
```

**Note**



The Nucleus WPA Supplicant is based on the Open Source wpa\_supplicant project. The original implementation contains detailed documentation of the wpa\_supplicant API functions. This Open Source documentation and the Control Interface of the wpa\_supplicant are useful for advanced users. The following is a link to the external developer documentation: [http://w1.fi/wpa\\_supplicant/wpa\\_supplicant-devel.pdf](http://w1.fi/wpa_supplicant/wpa_supplicant-devel.pdf). Chapter 3 of the external documentation covers the Control Interface of the wpa\_supplicant.

---

- [WPA\\_Supplicant\\_Reload\\_Config](#)
- [WPA\\_Supplicant\\_Get\\_Iface\\_Count](#)

## WPA\_Supplicant\_Reload\_Config

This function makes the WPA Supplicant reload its configuration file without re-starting the service. Most of Nucleus WPA Supplicant configuration is performed through its configuration file that it reads at run-time. This is a very important API that allows you to re-configure the Nucleus WPA Supplicant.

### Usage

```
STATUS WPA_Supplicant_Reload_Config (VOID);
```

### Return Values

- **NU\_SUCCESS**

Upon successful completion.

Otherwise, the routine returns an error code.

### Example

```
STATUS status;

/* Overwrite the Nucleus WPA Supplicant configuration file. */
...

/* Make Nucleus WPA Supplicant re-load its configuration file. */
status = WPA_Supplicant_Reload_Config();

if (status == NU_SUCCESS)
{
    /* Nucleus WPA Supplicant re-configuration successful. */
    ...
}

else
{
    /* Nucleus WPA Supplicant re-configuration not successful. */
    ...
}
```

### Related Topics

[Nucleus WPA Supplicant API Functions](#)

## WPA\_Supplicant\_Get\_Iface\_Count

This function returns the number of network interfaces being serviced by the Nucleus WPA Supplicant.

### Usage

```
INT WPA_Supplicant_Get_Iface_Count (VOID);
```

### Return Values

- The number of network interfaces registered with the Nucleus WPA Supplicant.

### Example

```
INT iface_count;  
  
/* Get the number of network interfaces. */  
iface_count = WPA_Supplicant_Get_Iface_Count();  
  
/* "iface_count" contains the number of interfaces. */
```

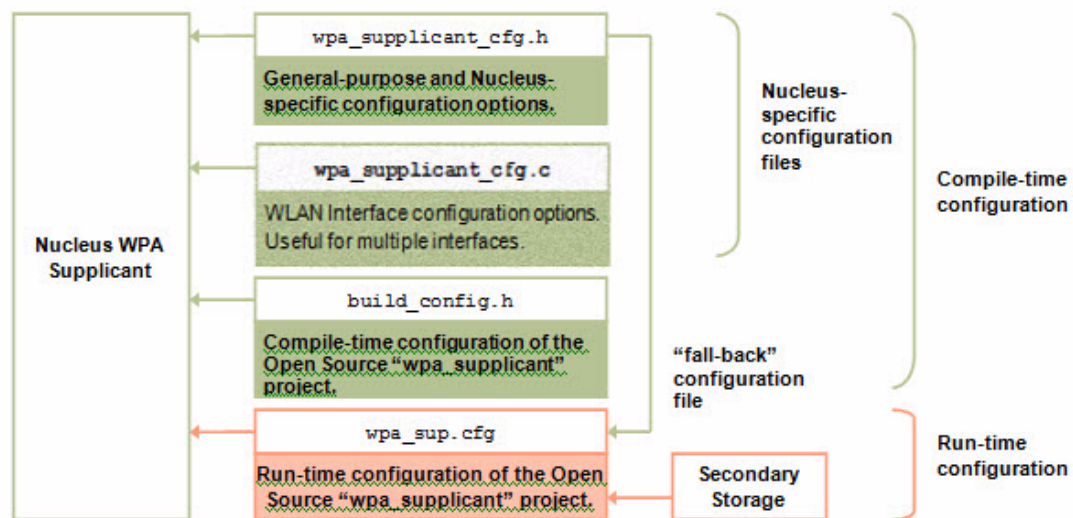
### Related Topics

[Nucleus WPA Supplicant API Functions](#)

## Nucleus WPA Supplicant Configuration

The Nucleus WPA Supplicant has several configuration files. This section explains the purpose, origin and configuration parameters of each of the configuration files. [Figure 18-2](#) provides an overview of the Nucleus WPA Supplicant configuration components.

**Figure 18-2. Nucleus WPA Supplicant Configuration Files**



It is important to note that the *wpa\_supplicant\_cfg.h*, *wpa\_supplicant\_cfg.c*, and *.metadata* files contain compile-time configuration options, while the *wpa\_sup.cfg* file is the run-time configuration file of the Nucleus WPA Supplicant.

The following sections further describe each of the configuration files shown in the diagram.

## .metadata

This file contains the most frequently used configuration settings of the Nucleus WPA Supplicant. In most cases, these are the only configuration options you need to modify. The location of this configuration file is *nucleus/os/networking/wpa\_supplicant/src/.metadata*. You do not usually directly modify this configuration file. Its default settings can be overridden from your platform configuration file.

The following configuration settings are defined in this file:

### nu.os.net.wpa\_supp.src.default\_drive

This configuration option defines the default file system drive which the Nucleus WPA Supplicant uses for reading the *wpa\_sup.cfg* file. The value of this option is an integer specifying the drive number. The value 0 maps to drive A, value 1 maps to drive B, and so on. Value 0 is usually the drive number for the ramdisk, value 1 is for removable disks, and value 2 is for fixed disks.

### nu.os.net.wpa\_supp.src.enable\_fallback\_conf

Nucleus WPA Supplicant supports a fall-back feature option. If the run-time *wpa\_sup.cfg* configuration file is not found, then it falls back to use an internally defined configuration file. To do this, it writes an internal copy of this file to disk. Set the macro to a value of 1 to enable the fall-back configuration file feature or set it to 0 to disable it. The default value of this macro is 1.

### nu.os.net.wpa\_supp.src.fallback\_conf\_data

This macro defines the contents of the fall-back configuration file. If the fall-back configuration file option is enabled, then make sure this macro contains a valid value. The file contents of this macro are equivalent to the *wpa\_supplicant.conf* file of the Open Source *wpa\_supplicant* project. You need to define the contents of this file as a single-line string in this configuration option. Be sure you separate new-lines with a new-line escape character and escape double-quote characters. The following escape characters must be used:

- `\\n` – Escaped new-line character.
- `\\` – Escaped double-quote character. If you override the value of an option in a platform configuration file, you might need to triple-escape the quote character by using `\\\`.

## **nu.os.net.wpa\_supp.src.ctrl\_iface**

Specify a unique path to the Control Interface. Set this macro to a blank string ("" ) if the control interface is disabled. This option is specified by the switch "-g" in the command-line client. Its default value is a blank string. An example of a value, non-blank value for this option is *"/var/run/wpa\_supplicant"*.

## **wpa\_sup.cfg**

The *wpa\_sup.cfg* file has the same format and serves the same purpose as the *wpa\_supplicant.conf* file of the Open Source “wpa\_supplicant” project. The Nucleus WPA Supplicant has an option for a “fall-back” configuration file (see [nu.os.net.wpa\\_supp.src.enable\\_fallback\\_conf](#) above).

This option allows you to define the default contents of the *wpa\_sup.cfg* configuration file, that are used if the file does not exist at the configured path. The Nucleus WPA Supplicant is configured by default to use the fall-back configuration file in the in-memory file system (IMF). The default contents of the file are defined by *nu.os.net.wpa\_supp.src.fallback\_conf\_data*.

You can also save this file in secondary storage. In that case, set the value of the *nu.os.net.wpa\_supp.src.default\_drive* setting to the drive containing this file.

The format of this file is the same as the one used by the Open Source wpa\_supplicant project. The format is described in the sample configuration file located in */os/networking/wpa\_supplicant/wpa\_supplicant/wpa\_supplicant.conf*.

## **wpa\_supplicant\_cfg.c**

Nucleus WPA Supplicant can service multiple WLAN Interfaces (if the underlying WLAN driver supports this). In this case, each WLAN Interface may require different configuration options. This configuration file defines the WPA Supplicant configuration for each WLAN Interface.

The *WPA\_Supplicant\_Ifaces\_Config* array in this file defines a list of configurations per WLAN Interface. One of the configurations (the last in the list) is usually the wildcard configuration, which matches all interface names. This list entry has a value of "\*" for the *ifname* field.

The following is a description of each configuration field for the configuration items of the *WPA\_Supplicant\_Ifaces\_Config* array:

**conf\_name:**

This option is specified by "-c" in the command-line client. This is a string that specifies the complete path to the wpa\_supplicant configuration file.

**ctrl\_interface:**

This option is specified by "-C" in the command-line client. This is a string that specifies the name of the global control interface path. It should only be used if the

"conf\_name" parameter is not specified; otherwise, this setting is specified in the *wpa\_sup.cfg* configuration file.

driver:

This option is specified by "-D" in the command-line client. It specifies the driver to use for the specified interface. The driver name for Nucleus-based interfaces is "nucleus\_wext".

driver\_param:

This option is specified by "-p" in the command-line client. It specifies an optional string which contains parameters to the driver. This should be set to NULL if not specified.

ifname:

This option is specified by "-i" in the command-line client. It should be the WLAN interface name which has been registered with Nucleus NET using the [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) API. It can also be set to a value of "\*" to match all interface names, but this wildcard entry should occur at the end of the list.

bridge\_ifname:

This option is specified by "-b" in the command-line client. It is a string specifying the bridge interface name.

---

**Note**

**IMPORTANT:** The list of configurations **MUST** be terminated by an item which contains all NULLs.

---

## wpa\_supplicant\_cfg.h

This file is located at */os/networking/wpa\_supplicant/wpa\_supplicant\_cfg.h*. It contains compile-time configuration options. Some options in this file are Nucleus-specific, such as task stack sizes and priorities. It also contains other options related to the WPA Supplicant.

### WPA\_SUPPLICANT\_INPUT\_BUFFER\_SIZE

This is the size of the incoming packets buffer in bytes. All incoming packets of the wpa\_supplicant are first copied into a buffer of this size and, then processed. The default value is 4000.

### WPA\_SUPPLICANT\_PARAM\_WAIT\_FOR\_MONITOR

If a control interface is specified, then set this macro to a non-zero value to make the wpa\_supplicant wait for a monitor program to connect to the control interface. This option is specified by "-W" in the command-line client. Its default value is 0.

### WPA\_SUPPLICANT\_PARAM\_DEBUG\_FILE\_PATH

This option allows sending debug output to a file instead of serial output. It should be set to a path/filename to the debug output file. This option is specified by the switch "-f" in the

command-line client. The default value of this macro is "wpa\_supplicant\_log.txt". Nucleus WPA Supplicant currently does not support sending debug output to a file so this option is unused.

### **WPA\_SUPPLICANT\_PARAM\_DEBUG\_LEVEL**

This is the debugging verbosity level. Possible values of this macro are listed here in increasing order of debug output:

- **MSG\_MSGDUMP**: Log packet dumps.
- **MSG\_DEBUG**: Log debug level messages.
- **MSG\_INFO**: Log informational messages.
- **MSG\_WARNING**: Log warning messages.
- **MSG\_ERROR**: Log error messages.

This option is specified by the switch "-d" and "-q" in the command line client. Its default value is **MSG\_MSGDUMP**.

### **WPA\_SUPPLICANT\_PARAM\_DEBUG\_SHOW\_KEYS**

This macro controls whether keys and passwords should be displayed in debug output. Enabling this option facilitates debugging, but it is a potential security threat if this option is enabled in production-grade systems. This option is specified by the switch "-K" in the command-line client. Set this macro to a non-zero value to enable this option. The default value of this option is 0.

### **WPA\_SUPPLICANT\_PARAM\_DEBUG\_SHOW\_TIMESTAMP**

This macro includes the timestamp in debug messages. To enable this option, set this macro to a non-zero value. This option is specified by the switch "-t" in the command-line client. Its default value is 0.

## **EAP-TLS Authentication Setup**

The Nucleus WPA Supplicant supports authentication using EAP-TLS. This section explains how to set up EAP-TLS on both the Ethernet switch and the Nucleus/target side. Before following these instructions, it is highly recommended that you run the Nucleus WPA Supplicant in another mode that is easier to configure, such as Open Mode or WPA-PSK.

## **Configuring the Nucleus Side for EAP-TLS**

Follow these instructions to configure Nucleus products and the Nucleus WPA Supplicant to use EAP-TLS. In the following instructions, you might need to update paths to certificate files in the configuration file according to your setup.



## Procedure

1. Generate the following certificate files:
  - *rand.der*
  - *root.der*
  - *srv\_cert.der*
  - *user\_cert.der*
  - *user\_cert\_key.der*
2. Place all the files except *srv\_cert.der* onto a storage disk that is accessible by the Nucleus WPA Supplicant.
3. Configure the *nu.os.net.wpa\_supp.src.fallback\_conf\_data* setting to use EAP-TLS. The following is an example value:

```
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1
network={
    ssid="mentor_wifi"
    key_mgmt=WPA-EAP
    eap=TLS
    pairwise=TKIP
    group=TKIP
    identity="supp_id"
    ca_cert="root.der"
    client_cert="user_cert.der"
    private_key="user_cert_key.der"
    private_key_passwd="whatever"
}
```

## Results

Your Nucleus WPA Supplicant is ready for use with EAP-TLS security. The next step is to configure your host/AP to use EAP-TLS.

# Configuration of a RADIUS Server and Ethernet Switch for EAP-TLS

This section consists of two sub-sections. The first sub-section [Setting Up a RADIUS Server](#) lists the generic setup instructions for configuring RADIUS Servers. The second sub-section [Configuring FreeRADIUS Server for Windows XP](#) lists the specific setup instructions for configuring the FreeRADIUS Server for Windows XP.

**Note**

The FreeRADIUS Server for Windows XP is the popular FreeBSD-based FreeRADIUS Server ported to MS Windows. It is available from <http://www.freeradius.net/>. This server is easy to install and configure and is recommended for use with the default demonstration of Nucleus WPA Supplicant.

---

## Setting Up a RADIUS Server

This section provides the generic configuration steps required to set up any RADIUS Server as the Authentication Server for WPA Supplicant authentication.

**Note**

For more detailed information on configuring your particular RADIUS Server, refer to its user guide.

---

### Procedure

1. Configure the Root and Server certificates on the RADIUS Server.

The Certification Authority whose Root certificate is being configured on the RADIUS Server must be the one that issued the client certificates for all your clients.

2. Configure the “Shared Secret” on your RADIUS Server.

This is a secret word shared between the Ethernet switch (acting as the Authenticator) and the RADIUS Server. It identifies the server to the switch.

3. If required, configure the RADIUS Server with the network address of your Authenticator (Ethernet Switch).
4. Configure the Users database of your RADIUS Server with the “Username” and “Password” (if required) of all the users that are allowed to authenticate to the Server.
5. Configure the EAP authentication method on the RADIUS Server.

This method can be any one of EAP-MD5, EAP-TLS, EAP-TTLS or EAP\_PEAP.

## Configuring FreeRADIUS Server for Windows XP

Download the FreeRADIUS Server for Windows XP from <http://www.freeradius.net/> and install it on your Windows XP testing machine.

Perform the steps in this section to configure the server.

## Procedure

1. Click **Start > Control Panel > Administrative Tools > Services** and verify the FreeRADIUS.net service has a status of “Started”.

If it is not started, start the service. To do this, right-click the FreeRADIUS icon in the System Tray and select the option **Start FreeRADIUS.net service**.

2. Right-click the FreeRADIUS icon in the System Tray and select **Edit EAP.conf**.

The file *EAP.conf* opens in the default editor.

3. Set the parameters as follows (Replace “C:/PATH-TO/” with the complete path to your Nucleus port):

```
default_eap_type = tls
private_key_password = whatever
private_key_file = C:/PATH-TO/demo/wpa_supPLICANT/certs/srv_cert.pem
certificate_file = C:/PATH-TO/demo/wpa_supPLICANT/certs/srv_cert.pem
CA_file = C:/PATH-TO/demo/wpa_supPLICANT/certs/root.pem
random_file = C:/PATH-TO/demo/wpa_supPLICANT/certs/rand.der
```

4. Edit the *<FreeRADIUS install directory>/etc/raddb/users* file and add the following lines to it:

```
supp_id    User-Password == "supp_pw"
Session-timeout = 500
```

5. Right-click the FreeRADIUS icon in the System Tray and select **Edit Radius Clients.conf**.

Search for the subnet block for your WLAN and copy/note its secret value.

For example, if the subnet is 192.168.0.0/24, the secret value is likely *testing123-1*.

This secret value is required during Wireless Ethernet Switch (AP) configuration - covered in the following section.

6. Right-click the FreeRADIUS icon in the System Tray and select **Restart FreeRADIUS.net service**.

---

### Note



If there is a firewall running on the test PC running the FreeRADIUS Server, it can interfere with FreeRADIUS communication. It is recommended to either disable the firewall during testing or bypass the FreeRADIUS Serverports 1812 and 1813.

---

## Configuring the WPA Supplicant-Capable Ethernet Switch

Use the steps provided in this section to configure a Wireless Ethernet Switch (AP) to work as a WPA Supplicant Authenticator. For specific switch configuration instructions, refer to the Ethernet Switch user guide.

Before you proceed, verify that your test machine (running the RADIUS server) is connected to the Wireless Ethernet Switch (AP) through a Wired Ethernet connection.

### Procedure

1. Select the WPA-Enterprise (or WPA-EAP) security mode on the switch.
2. Select the cipher type to be **TKIP**.
3. Configure the IP address of the RADIUS server that acts as the Authentication Server. This is the test machine IP address with RADIUS server installed.
4. Configure the shared secret.

This is a secret string the Ethernet Switch shares with the RADIUS Server. It identifies the switch to the RADIUS Server.

5. Save the settings and reboot the switch, if required.















# Chapter 19

## Point-To-Point Protocol (PPP)

---

### PPP Overview

The Point-to-Point Protocol (PPP) provides for communications between two network nodes connected via a point-to-point link. By utilizing PPP, Nucleus embedded devices are able to communicate with other host architectures such as Windows 95, Windows NT, UNIX, and so on.

PPP communications are typically over a serial device with one node dialing into the other. However, network mediums other than serial are not uncommon. PPP has even evolved to include network mediums that are not point-to-point. For example, PPP over Ethernet is commonly being deployed in residential gateways. The Nucleus PPP driver now includes support for PPPoE in addition to serial (HDLC) as the underlying physical medium.

PPP has commonly been used to deliver IPv4 based payloads, which typically contain UDP or TCP transmissions to and from the application. The addition of IPv6 as an additional option for Internet delivery makes it necessary to incorporate IPv6 capability in PPP as well, since it is a link protocol that provides the network addresses that the IP layer uses. Therefore, Nucleus PPP now includes support for IPv6, should it be necessary in the Nucleus NET application. See the [IPv6 Over PPP](#) section for more information.

Additionally, PPP link information can now be obtained and managed remotely via SNMP. This information includes which link options were successfully negotiated for an established link, which options will be used for subsequent connections, PPP user information, IP address information, and so on. See the [Using PPP with SNMP](#) section for more information.

### Required PPP Include File

To make all the PPP component APIs visible to an application, include the file *nu\_drivers.h* in your application, as in the following statement:

```
#include "/nucleus/os/include/drivers/nu_drivers.h"
```



#### Warning

In your applications, use only interfaces, structures, macros, and so on, that are documented within this and other Nucleus guides. There is no guarantee of future support or compatibility for any interface that is not documented.

## Nucleus PPP Driver Modules

The Nucleus PPP driver as shipped consists of three conceptual modules. The first, PPP, is the module that consists of the core PPP protocols. This module provides the heart of PPP and it is used no matter what the underlying network medium is. This module is discussed further in the [Nucleus PPP Module](#) section.

The second module is HDLC. This module contains the functionality necessary to communicate over true point-to-point links such as those supported by a UART or serial device. This is the type of link for which PPP was originally intended. For such devices, it is generally the case that the physical connection must be established before network communications can commence. In other words, a PPP client must dial up a PPP server. When communications are complete, the PPP client hangs up and the physical connection is broken. This module also contains the logic for driving a modem, and an API for managing modem and serial communications. The HDLC module is discussed further in the [Multilink Protocol](#) section.

The third module is the PPPoE module. PPPoE provides for PPP communications over an Ethernet network. Ethernet is not a point-to-point medium. However, the PPPoE module allows applications to establish a virtual circuit between a PPPoE Host (client) and a PPPoE Access Concentrator (server). Even though Ethernet is not a point-to-point medium, the services [NU\\_Dial\\_PPP\\_Server](#) and [NU\\_Wait\\_For\\_PPP\\_Client](#) must still be used to establish the virtual circuit. Also, a virtual PPP device is registered with Nucleus NET via the [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) API service. This module is discussed further in the [HDLC Interface](#) section.

## Applications

---

### Note

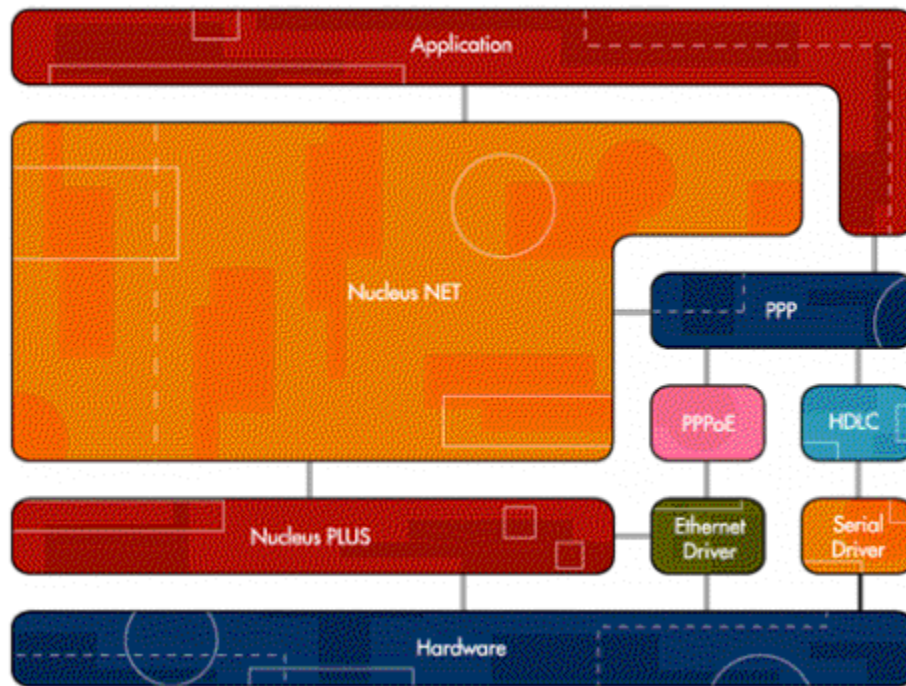


The PPP driver is transparent to the application layer, with two exceptions. The first exception is during system initialization, when the PPP device is registered with Nucleus NET via the [NU\\_Init\\_Net \(IPv4/IPv6\)](#) service. The second exception is that, unlike broadcast media such as Ethernet, the physical connection must be made before networking applications attempt to send TCP or UDP packets. The PPP API described in the [Nucleus PPP Module](#) section provides services for establishing and managing a dial-up connection. The details for registering an HDLC and a PPPoE device with Nucleus NET are discussed further [Multilink Protocol](#) and [HDLC Interface](#) sections.

---

[Figure 19-1](#) illustrates the relationship between the PPP, the modules that make up PPP, and the other Nucleus products.

**Figure 19-1. PPP Relationship with Other Modules and Other Nucleus Products**



## Nucleus PPP Module

PPP, being a driver-level protocol, is for the most part transparent to the application layer. However, before TCP/IP packets can be exchanged via PPP, the physical connection must be established. This is typically done via dial-up using a modem. Note that even when the physical network medium is always connected, as is the case with Ethernet, it is still necessary to use [NU\\_Dial\\_PPP\\_Server](#) if you are a client or [NU\\_Wait\\_For\\_PPP\\_Client](#) if you are a server in order to establish a connection. This chapter contains those API services that are used regardless of what physical medium is used.

## PPP Data Structures

This section provides details for data structures that are defined and used specifically with the Nucleus PPP API functions.

- [NU\\_PPP\\_OPTIONS](#)
- [NU\\_PPP\\_CFG](#)

## NU\_PPP\_OPTIONS

NU\_PPP\_OPTIONS is the typedef name for the ppp\_call\_options data structure. This structure is used to store PPP options.

```
typedef struct ppp_call_options
{
    UINT32 ppp_flags;
    CHAR    *ppp_num_to_dial;
    CHAR    *ppp_link;
    CHAR    *ppp_mp_link;
    UINT8    *ppp_local_ip4;
    UINT8    *ppp_remote_ip4;
    UINT8    *ppp_local_ip6;
    UINT8    *ppp_remote_ip6;
} PPP_OPTIONS, NU_PPP_OPTIONS;
```

The members of the structure are defined in [Table 19-1](#). Some members of this structure have been omitted as they are used internally.

**Table 19-1. NU\_PPP\_OPTIONS**

Member	Description
ppp_flags	Flags used for setting options. Valid option for this API is: NU_PPP_NO_IPV4 This disables the IPv4 negotiation for this link even if IPv4 is enabled in NET. This option is not used for <a href="#">NU_Set_PPP_Client_IP_Address</a> .

**Table 19-1. NU\_PPP\_OPTIONS (cont.)**

Member	Description
ppp_num_to_dial	Pointer to the phone number (or PPPoE service) to be dialed. NOTE: A '~' can be used in the phone number to insert a half second delay. If a direct cable is used for connection then null string can be passed. This option is not used for <a href="#">NU_Set_PPP_Client_IP_Address</a> and <a href="#">NU_Wait_For_PPP_Client</a> .
ppp_link	Pointer to the name of the PPP link to which this service should be applied.
ppp_mp_link	Pointer to the name of the Multilink Protocol virtual device with which to attach the above PPP link. This option is only valid if Multilink Protocol is enabled. For more information on this, refer to the <a href="#">Multilink Protocol</a> section. This option is not used for <a href="#">NU_Set_PPP_Client_IP_Address</a> and <a href="#">NU_Wait_For_PPP_Client</a> .
ppp_local_ip4	Pointer to an IPv4 address to be presented to the PPP server for use. The PPP server does not have to allow the client to use this address and in most cases will not allow its use. The most common use of this parameter is to use the address 0.0.0.0, which requests the PPP server to assign an address to the client. This option is not used for <a href="#">NU_Set_PPP_Client_IP_Address</a> .
ppp_remote_ip4	Pointer to the IPv4 address to be assigned to a calling client for the supplied link. This option is not used for <a href="#">NU_Dial_PPP_Server</a> and <a href="#">NU_Wait_For_PPP_Client</a> .
ppp_local_ip6	Pointer to an IPv6 address to be presented to the PPP server for use. The PPP server does not have to allow the client to use this address and in most cases will not allow its use. The most common use of this parameter is to use the address 0.0.0.0.0.0.0, which requests the PPP server to assign an address to the client. This option is not used for <a href="#">NU_Set_PPP_Client_IP_Address</a> .
ppp_remote_ip6	Pointer to the IPv6 address to be assigned to a calling client for the supplied link. This option is only valid if IPv6 is enabled. This option is not used for <a href="#">NU_Dial_PPP_Server</a> and <a href="#">NU_Wait_For_PPP_Client</a> .

## Related Topics

[PPP Data Structures](#)

[NU\\_Dial\\_PPP\\_Server](#)

[NU\\_Set\\_PPP\\_Client\\_IP\\_Address](#)

[NU\\_Wait\\_For\\_PPP\\_Client](#)

## NU\_PPP\_CFG

NU\_PPP\_CFG is the typedef name for the `ppp_cfg_struct` data structure. This structure is defined in *include/drivers/ppp\_defs.h* and should be allocated in the application.

```
typedef struct ppp_cfg_struct
{
    UINT16  default_auth_protocol;
    UINT16  default_mru;
    UINT32  use_flags;
    UINT32  default_accm;
    UINT32  default_fcs_size;
    UINT32  num_dns_servers;
    UINT8   use_128_bit_encryption;
    UINT8   use_56_bit_encryption;
    UINT8   use_40_bit_encryption;
    UINT8   require_encryption;
    UINT16  mp_mrru;
} PPP_CFG, NU_PPP_CFG;
```

The members of the structure are defined in [Table 19-2](#). Some members of this structure have been omitted as they are used internally.

**Table 19-2. NU\_PPP\_CFG**

Member	Description
default_auth_protocol	Gets the default authentication protocol to be used for this link. The default protocol is negotiated first when a client connects. Types include PAP, CHAP, MSCHAPv1, and MSCHAPv2: <ul style="list-style-type: none"><li>• NU_PPP_CHAP_MS2_PROTOCOL (0x81)</li><li>• NU_PPP_CHAP_MS1_PROTOCOL (0x80)</li><li>• NU_PPP_CHAP_PROTOCOL (0xC223)</li><li>• NU_PPP_PAP_PROTOCOL (0xC023)</li></ul> Any other value disables authentication negotiation.
default_mru	Gets the maximum number of characters this device can receive in one packet. If this is set to 0, the default MRU of 1500 octets is used and the MRU LCP option is not negotiated for the local side.
use_flags	Flags used for enabling or disabling options. Refer to <a href="#">Table 19-3</a> for valid options.
default_accm	Gets the default Asynchronous Control Character Map for the local side of the link.
default_fcs_size	Gets the default Frame Check Size of the link.



**Table 19-2. NU\_PPP\_CFG (cont.)**

Member	Description
num_dns_servers	Request primary and secondary DNS server addresses for use by this link, depending on the value assigned. A value of 1 specifies that only a primary DNS server is requested. A value of 2 specifies that both a primary and secondary DNS server are requested. No DNS server addresses are requested if any other value is assigned.
use_128_bit_encryption	Enable or disable the use of 128-bit MPPE encryption on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
use_56_bit_encryption	Enable or disable the use of 56-bit MPPE encryption on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
use_40_bit_encryption	Enable or disable the use of 40-bit MPPE encryption on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
require_encryption	Enable or disable the use of MPPE Encryption for this link. Possible values for this option are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
mp_mrru	Gets the maximum received reconstructed unit size of an MP interface.

The following flags can be used for the use\_flags option:

**Table 19-3. use\_flags Options for NU\_PPP\_CFG**

Flags	Description
PPP_FLAG_ACCM	Enables or disables the negotiation of the ACCM option.
PPP_FLAG_PFC	Enables or disables protocol field compression as defined in RFC1661.
PPP_FLAG_ACC	Enables or disables address field compression as defined in RFC1661.
PPP_FLAG_MAGIC	Enables or disables magic number negotiation as defined in RFC1661.
PPP_MP_SHORT_SEQ	Enables or disables the use of short sequence number header over MP links. Note that all links associated with a particular MP interface either use short sequence number headers or they all would not use it.

## Related Topics

[PPP Data Structures](#)

[NU\\_Validate\\_Link\\_Options](#)

[NU\\_Set\\_PPP\\_Link\\_Options](#)

[NU\\_PPP\\_Get\\_Default\\_Options](#)

[NU\\_Get\\_PPP\\_Link\\_Options](#)

## PPP Services

All PPP services take the name of the device as a parameter. The name is chosen at the time that the device is registered with Nucleus NET via the [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) service. Thereafter, the same name must be used anytime a PPP API service is called. By naming the devices and supplying a name to each service, the ability to support multiple PPP links is obtained.

The following is a list of the PPP services and a brief description of each.

- [NU\\_Dial\\_PPP\\_Server](#)
- [NU\\_PPP\\_Hangup](#)
- [NU\\_Set\\_PPP\\_Login](#)
- [NU\\_Add\\_PPP\\_User](#)
- [NU\\_Remove\\_PPP\\_User](#)
- [NU\\_Set\\_PPP\\_Client\\_IP\\_Address](#)
- [NU\\_PPP\\_Still\\_Connected](#)
- [NU\\_Wait\\_For\\_PPP\\_Client](#)
- [NU\\_PPP\\_Get\\_Default\\_Options](#)
- [NU\\_Validate\\_Link\\_Options](#)
- [NU\\_Get\\_PPP\\_Link\\_Options](#)
- [NU\\_Get\\_PPP\\_Link\\_Option](#)
- [NU\\_Set\\_PPP\\_Link\\_Options](#)
- [NU\\_Set\\_PPP\\_Link\\_Option](#)
- [NU\\_PPP\\_Last\\_Activity\\_Time](#)
- [NU\\_PPP\\_Negotiation\\_Timeout](#)
- [NU\\_Obtain\\_PPP\\_Connection\\_Status](#)
- [NU\\_Abort\\_Wait\\_For\\_PPP\\_Client](#)

- [NU\\_PPP\\_Abort](#)
- [NU\\_PPP\\_Abort\\_Connection](#)

## NU\_Dial\_PPP\_Server

This service provides the facility to pass-in various options for connecting with a PPP server.

### Usage

```
STATUS NU_Dial_PPP_Server (NU_PPP_OPTIONS *ppp_options);
```

### Arguments

- `ppp_options`  
Pointer to the [NU\\_PPP\\_OPTIONS](#) structure for passing in different options.

### Return Values

- `NU_SUCCESS`  
A connection was made.
- `NU_INVALID_POINTER`  
The number pointer is `NU_NULL`.
- `NU_NO_MODEM`  
A modem on the specific link was not found. Either it is not powered, not connected, or simply not present at all.
- `NU_NO_CONNECT`  
The connection was not made.
- `NU_NO_CARRIER`  
The receiver did not send a carrier.
- `NU_BUSY`  
The number dialed was busy.
- `NU_LCP_FAILED`  
PPP could not successfully negotiate the options for the link.
- `NU_LOGIN_FAILED`  
Either the ID or password used for logging into the remote system is wrong or there is no valid account with the remote system.
- `NU_NCP_FAILED`  
PPP could not negotiate an IP address.
- The link name supplied is not a valid PPP device.

### Description

For the list of options, refer to the [NU\\_PPP\\_OPTIONS](#) data structure. Before connecting to a PPP server, a call to [NU\\_Set\\_PPP\\_Login](#) must be made in order for a successful connection to

be established. The only exception would be if it were known in advance that the PPP server will not require authentication.

### Example

```
NU_PPP_OPTIONS    ppp_options;
STATUS            status;
UINT8             local_ip4 = {192,168,0,50};
UINT8             local_ip6 = {0,0,0,0,0,0,0,2};
...

/* Number of the PPP server to dial */
ppp_options.ppp_num_to_dial = "5712234";

/* Link name on which the connection is to be established */
ppp_options.ppp_link = "ppp0";

/* Local IPv4 address which we want to negotiate with the PPP
 * server to use for this end of the link
 */
ppp_options.ppp_local_ip4 = local_ip4;

/* Local IPv6 address which we want to negotiate with the PPP server
 * to use for this end of the link.  If the PPP server agrees
 * then the local IPv6 address would become FE80::2.
 */
ppp_options.ppp_local_ip6 = local_ip6;

/* Dial the PPP server as set by the options above. */
status = NU_Dial_PPP_Server (&ppp_options);

...
```

### Related Topics

[PPP Services](#)

[NU\\_Set\\_PPP\\_Login](#)

[PPP Data Structures](#)

[NU\\_PPP\\_OPTIONS](#)

## NU\_PPP\_Hangup

This service terminates the PPP link and commands the modem to hang up.

### Usage

```
STATUS NU_PPP_Hangup (UINT8 mode,  
                     CHAR *link_name);
```

### Arguments

- mode

How to terminate the link.

NU\_NO\_FORCE

Begin the hang up process, safely disconnecting TCP sockets connections.

NU\_FORCE

‘Demand’ the hang up process without any further transmissions to the wire. This aborts open TCP connections.

NU\_NO\_FORCE|NU\_WAIT\_FOR\_HANGUP

Start the hang up process and wait until it is complete.

- link\_name

Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- NU\_TRUE

The connection has been closed.

- NU\_FALSE

The link could not be closed because the PPP link is currently being used. This is only returned if the NU\_NO\_FORCE option is used.

- NU\_INVALID\_LINK

The link name supplied is not a valid PPP device.

### Description

Since it is possible that other tasks may be using the network, there are two options for terminating the link. By setting the NU\_FORCE option disconnection is started at the PPP layer, essentially shutting down the device without allowing upper layers to negotiate termination with their peers. Setting the NU\_NO\_FORCE option merely starts the shut down process, allowing upper layers to properly negotiate the termination of their connections before negotiating termination of its own connection. NU\_FORCE mode can be likened to a ‘demand’, while NU\_NO\_FORCE mode is more like a request.

Using either hangup mode, the function always returns NU\_FALSE, signifying that the hang up process is not complete and the device is not ready to initiate another connection. The function

simply starts the event-driven disconnection process and returns immediately so that the calling thread can perform other duties. If the task needs to determine that the modem has indeed disconnected, it can then call the function again as many times as necessary.

If there are no other duties to perform while the modem is disconnecting, the `NU_WAIT_FOR_HANGUP` flag can be used with one of the two `FORCE` modes. The task then suspends until the disconnection is complete, and the return value is `NU_TRUE`.

### Example

```
STATUS          status;

/* Loop forever. */
for(;;)
{
    /* If an external PPP hang-up request was made
     * and a PPP connection is currently established
     */
    if((hangup_request == NU_TRUE) &&
        (NU_PPP_Still_Connected("ppp0") == NU_TRUE))
    {
        /* Force PPP hang-up. */
        NU_PPP_Hangup(NU_FORCE, "ppp0");
    }
}
```

### Related Topics

[PPP Services](#)

## NU\_Set\_PPP\_Login

This service sets the ID and password to be used when dialing into a service provider.

### Usage

```
STATUS NU_Set_PPP_Login (CHAR id[PPP_MAX_ID_LENGTH],  
                        CHAR pw[PPP_MAX_PW_LENGTH],  
                        CHAR *link_name);
```

### Arguments

- **id**  
Pointer to the ID or name to be used when logging on to a remote system.
- **pw**  
Pointer to the password to be used when logging on to a remote system.
- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied, or if a Multilink PPP call is to be established, then it is the pointer to the name of the MP virtual device. Refer to the [Multilink Protocol](#) section for more details.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The ID and password were set.

### Description

This must be set before an attempt to connect is made, via the [NU\\_Dial\\_PPP\\_Server](#) service. An ID or password longer than the preset max is truncated at the max length. The preset max is set in *include/drivers/ppp\_cfg.h*; see [PPP Configuration Options](#). Once the pair is set, they remain set, per link, until another call to [NU\\_Set\\_PPP\\_Login](#) is made.

### Example

```
STATUS          status;  
UINT8          local_ip4 = {192,168,0,50};  
...  
  
/* Initialize the modem. */  
DEMOI_InitModem("ppp0");  
  
/* Set the login and password. */  
NU_Set_PPP_Login("fred", "fish", "ppp0");
```



## Related Topics

[PPP Services](#)

[NU\\_Dial\\_PPP\\_Server](#)

[NU\\_Set\\_PPP\\_Login](#)

## NU\_Add\_PPP\_User

This service adds a remote PPP user to the PPP Security Secrets MIB table if the MIB is built into the system.

### Usage

```
STATUS NU_Add_PPP_User (CHAR *username,  
                        CHAR *password);
```

### Arguments

- **username**  
Pointer to the name of a remote user to be authenticated on dial-in.
- **password**  
Pointer to the password checked when the user is authenticated.

### Return Values

- **UM\_USER\_UNKNOWN**  
The user could not be added to the MIB table.
- **NU\_SUCCESS**  
The user has been added to the MIB table.

### Description

The [NU\\_Add\\_PPP\\_User](#) function must be called immediately after adding the user via the standard UM (User Management) call, [UM\\_Add\\_User](#), which is defined in Nucleus NET. If the PPP Security Secrets MIB is not built into the system, then this function is unnecessary.

---

#### Note



The Security Secrets MIB can be enabled or disabled in *include/drivers/ppp\_cfg.h*.

---

### Example

```
#if (PPP_ENABLE_UM_DATABASE == NU_TRUE)  
    /* Add some PPP users to the UM database. */  
    UM_Add_User("fred", "fish", UM_PPP, 0);  
    UM_Add_User("joe", "blow", UM_PPP, 0);  
    UM_Add_User("jon", "doe", UM_PPP, 0);  
#endif  
  
#if (INCLUDE_SEC_MIB == NU_TRUE)  
    /* Also add them to the PPP MIB. */  
    NU_Add_PPP_User("fred", "fish");  
    NU_Add_PPP_User("joe", "blow");  
    NU_Add_PPP_User("jon", "doe");  
#endif
```

## Related Topics

[PPP Services](#)

[UM\\_Add\\_User](#)

[NU\\_Add\\_PPP\\_User](#)

## NU\_Remove\_PPP\_User

This service removes a remote PPP user from the PPP Security Secrets MIB table if the MIB is built into the system. It must be called before deleting the user via the standard UM (User Management) call, [UM\\_Del\\_User](#), which is defined in Nucleus NET. If the PPP Security Secrets MIB is not built into the system, then this function is unnecessary.

---

### Note



The Security Secrets MIB can be enabled or disabled in *include/drivers/ppp\_cfg.h*.

---

### Usage

```
STATUS NU_Remove_PPP_User (CHAR *username);
```

### Arguments

- **username**  
Pointer to the name of the user to be removed from the MIB table.

### Return Values

- **NU\_SUCCESS**  
The user has been removed from the MIB table.
- **UM\_USER\_UNKNOWN**  
The user was not found in the MIB table.

### Example

```
#if (INCLUDE_SEC_MIB == NU_TRUE)
/* Remove users from the PPP MIB. */
NU_Remove_PPP_User("fred");
NU_Remove_PPP_User("joe");
NU_Remove_PPP_User("jon");
#endif

#if (PPP_ENABLE_UM_DATABASE == NU_TRUE)
/* Also remove them from the UM database. */
UM_Del_User("fred");
UM_Del_User("joe");
UM_Del_User("jon");
#endif
```

### Related Topics

[PPP Services](#)

[UM\\_Del\\_User](#)

## NU\_Set\_PPP\_Client\_IP\_Address

This service sets the IP addresses (IPv4 and IPv6) to be assigned to a calling client for a particular PPP link. This is only used when a call to [NU\\_Wait\\_For\\_PPP\\_Client](#) is made. Once the IP address is set, it remains set, per link, until another call to [NU\\_Set\\_PPP\\_Client\\_IP\\_Address](#) is made.

### Usage

```
STATUS NU_Set_PPP_Client_IP_Address (NU_PPP_OPTIONS *ppp_options);
```

### Arguments

- `ppp_options`  
Pointer to the [NU\\_PPP\\_OPTIONS](#) structure to use for this service.

### Return Values

- `NU_SUCCESS`  
The address was taken.
- `NU_INVALID_ADDRESS`  
The address to be assigned is the same as the one NET is using.
- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.

### Example

```
NU_PPP_OPTIONS    ppp_options;
STATUS            status;
UINT8            remote_ip4 = {192,168,0,60};
UINT8            remote_ip6 = {0,0,0,0,0,0,0,3};
...

/* PPP link name on which a call is received */
ppp_options.ppp_link = "ppp0";

/* Remote IPv4 address which we want to set for the calling
 * client
 */
ppp_options.ppp_remote_ip4 = remote_ip4;

/* Remote IPv6 address which we want to set for the calling
 * client. The IPv6 address for the client would be FE80::3.
 */
ppp_options.ppp_remote_ip6 = remote_ip6;

/* Set the IP address that is assigned to the remote
 * client.
 */
NU_Set_PPP_Client_IP_Address(&ppp_options);

...
```

## Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_Wait\\_For\\_PPP\\_Client](#)

[NU\\_PPP\\_OPTIONS](#)

## NU\_PPP\_Still\_Connected

This service is similar to the HDLC [NU\\_Carrier](#) service, but instead of returning the state of the modem, it returns the logical state of the link.

### Usage

```
STATUS NU_PPP_Still_Connected (CHAR *link_name);
```

### Arguments

- `link_name`  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- `NU_TRUE`  
The link is still open.
- `NU_FALSE`  
The link is closed.
- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.

### Example

```
STATUS status;
NU_PPP_OPTIONS ppp_options;
UINT8 local_ip4 = {192,168,1,98};
UINT8 remote_ip4 = {192,168,1,97};

...

/* PPP link name on which a call is received */
ppp_options.ppp_link = "ppp0";

/* Remote IPv4 address which we want to set for the calling client
*/
ppp_options.ppp_remote_ip4 = remote_ip4;

/* Local IPv4 address which we want to set for this PPP server
*/
ppp_options.ppp_local_ip4 = local_ip4;

/* Set the IP address that is assigned to the remote client.
*/
NU_Set_PPP_Client_IP_Address(&ppp_options);

/* Wait for a client to call. */
NU_Wait_For_PPP_Client(&ppp_options);

/* Wait until the session is ended before we continue. */
while ((NU_PPP_Still_Connected("ppp0") == NU_TRUE))
    NU_Sleep(TICKS_PER_SECOND);
```

```
/* Force hang-up. */  
NU_PPP_Hangup(NU_FORCE, "ppp0");
```

## Related Topics

[PPP Services](#)

[NU\\_Carrier](#)



## NU\_Wait\_For\_PPP\_Client

This service puts the modem in answer mode and waits for a client to call.

### Usage

```
STATUS NU_Wait_For_PPP_Client (NU_PPP_OPTIONS *ppp_options);
```

### Arguments

- `ppp_options`  
Pointer to the [NU\\_PPP\\_OPTIONS](#) structure.

### Return Values

- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.
- `NU_PPP_ATTEMPT_ABORTED`  
Wait for client was aborted by another task through a call to [NU\\_Abort\\_Wait\\_For\\_PPP\\_Client](#).
- `NU_SUCCESS`  
A client has successfully connected.

If `NU_PPP_RETURN_ERROR` flag was set during the call, the following additional errors may be returned:

- `NU_NEGOTIATION_TIMEOUT`  
PPP negotiation timed out.
- `NU_NEGOTIATION_ABORTED`  
PPP negotiation was aborted.
- `NU_LCP_FAILED`  
LCP negotiations failed.
- `NU_AUTH_FAILED`  
Authentication failed.
- `NU_NCP_FAILED`  
NCP negotiations failed.

### Description

Once the modems have connected, PPP begins negotiating the link and authenticating the client. When control returns from this function, a client has successfully negotiated the PPP link and is currently connected to the system, assuming the return status is `NU_SUCCESS`. Once a client

starts a PPP session, the IP addresses are added to Nucleus NET's routing table. Note that a call to [NU\\_Set\\_PPP\\_Client\\_IP\\_Address](#) must be made before a client can be connected.

### Example

```
NU_PPP_OPTIONS    ppp_options;
STATUS            status;
UINT8             local_ip4 = {192,168,0,1};
UINT8             local_ip6 = {0,0,0,0,0,0,0,1};
UINT8             remote_ip4 = {192,168,0,60};
UINT8             remote_ip6 = {0,0,0,0,0,0,0,3};

...

/* PPP link name on which a call is received. */
ppp_options.ppp_link = "ppp0";

/* Local IPv4 address which we want to set for this PPP
 * server
 */
ppp_options.ppp_local_ip4 = local_ip4;

/* Local IPv6 address which we want to set for this PPP
 * server. The IPv6 address for the server would be FE80::1.
 */
ppp_options.ppp_local_ip6 = local_ip6;

/* Remote IPv4 address which we want to set for the calling
 * client
 */
ppp_options.ppp_remote_ip4 = remote_ip4;

/* Remote IPv6 address which we want to set for the calling
 * client. The IPv6 address for the client would be FE80::3.
 */
ppp_options.ppp_remote_ip6 = remote_ip6;

/* Set the IP addresses that are assigned to the remote
 * client.
 */
NU_Set_PPP_Client_IP_Address(&ppp_options);

/* Start waiting for the client. */
status = NU_Wait_For_PPP_Client(&ppp_options);

...
```

### Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_Set\\_PPP\\_Client\\_IP\\_Address](#)

[NU\\_PPP\\_OPTIONS](#)

## NU\_PPP\_Get\_Default\_Options

This service gets the default configuration values for the [NU\\_PPP\\_CFG](#) structure. This routine provides a quick way to populate the PPP configuration structure with default values.

### Usage

```
VOID NU_PPP_Get_Default_Options (NU_PPP_CFG *ppp_cfg);
```

### Arguments

- `ppp_cfg`  
Pointer to a [NU\\_PPP\\_CFG](#) data structure.

### Return Values

- None

### Example

```
NU_PPP_CFG    ppp_cfg;  
  
/* Set the default link options for the PPP link "ppp0". */  
NU_PPP_Get_Default_Options(&ppp_cfg);
```

### Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_PPP\\_CFG](#)

## NU\_Validate\_Link\_Options

This service validates the configuration values for the [NU\\_PPP\\_CFG](#) structure.

### Usage

```
STATUS NU_Validate_Link_Options (NU_PPP_CFG *ppp_cfg);
```

### Arguments

- `ppp_cfg`  
Pointer to a [NU\\_PPP\\_CFG](#) data structure.

### Return Values

- `NU_PPP_INVALID_PARAMS`  
The parameter is not valid.
- `NU_INVALID_DEFAULT_MRU`  
Invalid MRU was specified.
- `NU_INVALID_DEFAULT_PROTOCOL`  
Invalid Default Authentication Protocol was specified.
- `NU_INVALID_NUM_DNS_SERVERS`  
Invalid number of DNS Servers specified.
- `NU_INVALID_PF_COMPRESSION`  
Invalid Protocol Field Compression specified.
- `NU_INVALID_AC_COMPRESSION`  
Invalid Address Control Compression specified.
- `NU_INVALID_USE_MAGIC_NUMBER`  
Invalid Magic Number specified.
- `NU_INVALID_REQUIRE_ENCRYPTION`  
Invalid value for `require_encryption` specified.
- `NU_INVALID_40BIT_ENCRYPTION`  
Invalid value for `use_40_bit_encryption` specified.
- `NU_INVALID_56BIT_ENCRYPTION`  
Invalid value for `use_56_bit_encryption` specified.
- `NU_INVALID_128BIT_ENCRYPTION`  
Invalid value for `use_128_bit_encryption` specified.
- `NU_INVALID_USE_ACCM`  
Invalid value for `use_accm` specified.

- NU\_SUCCESS

All the PPP configurations are valid.

### Example

```
NU_PPP_CFG    ppp_cfg;
STATUS        status;

/* Get the default link options for the PPP link "ppp0". */
NU_PPP_Get_Default_Options(&ppp_cfg);

/* Update the default link options. */
ppp_cfg.use_accm = NU_FALSE;

status = NU_Validate_Link_Options(&ppp_cfg);

if (status == NU_SUCCESS)
{
    /* Perform further operations. */
}
```

### Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_PPP\\_CFG](#)

## NU\_Get\_PPP\_Link\_Options

This service gets the configurable LCP and NCP options for the PPP link.

### Usage

```
STATUS NU_Get_PPP_Link_Options (CHAR          *link_name,  
                                NU_PPP_CFG *ppp_cfg);
```

### Arguments

- `link_name`  
The name of the PPP link for which the options are returned.
- `ppp_cfg`  
Pointer to a [NU\\_PPP\\_CFG](#) data structure.

### Return Values

- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.
- `NU_SUCCESS`  
The options were successfully assigned.

### Description

All options within the [NU\\_PPP\\_CFG](#) structure are assigned a value. An example use of this function can be seen in the following example. The default options are defined by constants in *include/drivers/ppp\_cfg.h*.

### Example

```
NU_PPP_CFG    ppp_cfg;  
STATUS        status;  
...  
  
/* Get link options for the PPP link "ppp0". */  
status = NU_Get_PPP_Link_Options("ppp0", &ppp_cfg);  
  
if(status == NU_SUCCESS)  
{  
    /* Check if address protocol field compression is  
     * set or not.  
     */  
    if(ppp_cfg.use_flags & PPP_FLAG_PFC)  
    {  
        /* Protocol field compression is currently set  
         * on the PPP link.  
         */  
    }  
}
```

## Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_PPP\\_CFG](#)

## NU\_Get\_PPP\_Link\_Option

This service gets the configurable LCP, NCP, and CCP options for the PPP link. Instead of getting all of the PPP options, this routine can be used to get the individual PPP options.

### Usage

```
STATUS NU_Get_PPP_Link_Option (CHAR *link_name,  
                               INT  optname,  
                               VOID *optval,  
                               INT  *optlen);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link for which the options are returned.
- **optname**  
The name of the option to get. Refer to [Table 19-4](#) for valid options.

**Table 19-4. NU\_Get\_PPP\_Link\_Option Options**

Option	Description
PPP_DEFAULT_ACCM	(UINT32) Default Asynchronous Control Character Map for the local side of the link.
PPP_DEFAULT_AUTH_PROTOCOL	(UINT16) The default authentication protocol to be used for this link. The default protocol is negotiated first when a client connects. Types include PAP, CHAP, MSCHAPv1, and MSCHAPv2: <ul style="list-style-type: none"><li>• NU_PPP_CHAP_MS2_PROTOCOL (0x81)</li><li>• NU_PPP_CHAP_MS1_PROTOCOL (0x80)</li><li>• NU_PPP_CHAP_PROTOCOL (0xC223)</li><li>• NU_PPP_PAP_PROTOCOL (0xC023)</li></ul>
PPP_DEFAULT_FCS	(UINT32) Gets the default Frame Check Size of the link.
PPP_DEFAULT_MRU	(UINT16) Maximum number of characters this device can receive in one packet.
PPP_NUM_OF_DNS_SERVERS	(UINT32) Request primary and secondary DNS server addresses for use by this link, depending on the value assigned. A value of 1 specifies that only a primary DNS server is requested. A value of 2 specifies that both a primary and secondary DNS server are requested. No DNS server addresses are requested if any other value is assigned.
PPP_DEFAULT_MP_MRRU	(UINT16) Maximum receive reconstructed unit size of an MP interface.



**Table 19-4. NU\_Get\_PPP\_Link\_Option Options (cont.)**

Option	Description
PPP_ENABLE_128BIT_ENCRYPTION	(UINT8) Whether 128-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
PPP_ENABLE_56BIT_ENCRYPTION	(UINT8) Whether 56-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
PPP_ENABLE_40BIT_ENCRYPTION	(UINT8) Whether 40-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
PPP_ENABLE_ENCRYPTION_REQUIRED	(UINT8) Whether CCP negotiation is initiated for this link. Possible values for this option are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>
PPP_DEFAULT_FLAGS	(UINT32) Flags used for enabling or disabling options Refer to Table 19-3, under the <a href="#">NU_PPP_CFG</a> data structure description for valid options.

- **optval**  
A pointer to the location where the option value is written.
- **optlen**  
The size in bytes of the location pointed by optval. In case of invalid length the optlen is updated to reflect the required length.

### Return Values

- **NU\_PPP\_ENTRY\_NOT\_FOUND**  
A valid entry was not found.
- **NU\_PPP\_INVALID\_OPTION**  
The option specified is not valid.
- **NU\_PPP\_INVALID\_PARAMS**  
Either optlen or optval parameter is invalid.
- **NU\_SUCCESS**  
The options were successfully retrieved.

### Example

```
UINT32      ppp_flags;
```

```
STATUS      status;
...

/* Get link options for the PPP link "ppp0". */
status = NU_Get_PPP_Link_Option("ppp0", PPP_DEFAULT_FLAGS,
                                &ppp_flags, &sizeof(UINT32));

if(status == NU_SUCCESS)
{
    /* Check if address protocol field compression is
     * set.
     */
    if(ppp_flags & PPP_FLAG_PFC)
    {
        /* Protocol field compression is currently set
         * on the PPP link.
         */
    }
}
```

## Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_Get\\_PPP\\_Link\\_Options](#)

[NU\\_PPP\\_CFG](#)

## NU\_Set\_PPP\_Link\_Options

This service sets the configurable LCP and NCP options for the PPP link.

### Usage

```
STATUS NU_Set_PPP_Link_Options (CHAR          *link_name,  
                                NU_PPP_CFG *ppp_cfg);
```

### Arguments

- `link_name`  
Pointer to the name of the PPP link for which these options are assigned.
- `ppp_cfg`  
Pointer to a [NU\\_PPP\\_CFG](#) data structure.  
Each member of the structure should be defined before calling this function.

### Return Values

- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.
- `NU_SUCCESS`  
The options were successfully assigned.

### Description

In most cases it is not necessary to call this function. The default options used by PPP generally work. If it is necessary to call this service, all options within the [NU\\_PPP\\_CFG](#) structure should be assigned a value, because all values are assigned to the link unless otherwise noted. This service must be called after device initialization and should be called before a PPP connection attempt is made. An example use of this function can be seen in `HDLC_Initialize`, just after the call to `PPP_Initialize`. The default options are defined by constants in *include/drivers/ppp\_cfg.h*.

#### Note



If you are only setting one set of options, the other set needs to be given the parameter `NU_NULL`.

### Example

```
NU_PPP_CFG  ppp_cfg;  
STATUS      status;  
...  
  
/* Get link options for the PPP link "ppp0". */  
status = NU_Get_PPP_Link_Options("ppp0", &ppp_cfg);  
  
if(status == NU_SUCCESS)  
{
```

```
/* Set default authentication protocol to CHAP. */
ppp_cfg.default_auth_protocol = NU_PPP_CHAP_PROTOCOL;

/* Update the PPP link options. */
status = NU_Set_PPP_Link_Options("ppp0", &ppp_cfg);

if(status == NU_SUCCESS)
{
    /* Link options updated successfully. */
}
}
```

## Related Topics

[PPP Services](#)

[PPP Data Structures](#)

[NU\\_PPP\\_CFG](#)

## NU\_Set\_PPP\_Link\_Option

This service sets the configurable LCP, NCP, and CCP options for the PPP link. This service must be called after device initialization and should be called before a PPP connection attempt is made.

### Usage

```
STATUS NU_Set_PPP_Link_Option (CHAR *link_name,
                               INT  optname,
                               VOID *optval,
                               INT  optlen);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link for which these options are assigned.
- **optname**  
The name of the option to set. Refer to [Table 19-5](#) for valid options.

**Table 19-5. NU\_Set\_PPP\_Link\_Option Options**

Option	Description
PPP_DEFAULT_ACCM	(UINT32) Default Asynchronous Control Character Map for the local side of the link.
PPP_DEFAULT_AUTH_PROTOCOL	(UINT16) The default authentication protocol to be used for this link. The default protocol is negotiated first when a client connects. Types include PAP, CHAP, MSCHAPv1, and MSCHAPv2: <ul style="list-style-type: none"> <li>• NU_PPP_CHAP_MS2_PROTOCOL (0x81)</li> <li>• NU_PPP_CHAP_MS1_PROTOCOL (0x80)</li> <li>• NU_PPP_CHAP_PROTOCOL (0xC223)</li> <li>• NU_PPP_PAP_PROTOCOL (0xC023)</li> </ul>
PPP_DEFAULT_FCS	(UINT32) Gets the default Frame Check Size of the link.
PPP_DEFAULT_MRU	(UINT16) Maximum number of characters this device can receive in one packet.
PPP_NUM_OF_DNS_SERVERS	(UINT32) Request primary and secondary DNS server addresses for use by this link, depending on the value assigned. A value of 1 specifies that only a primary DNS server will be requested. A value of 2 specifies that both a primary and secondary DNS server will be requested. No DNS server addresses are requested if any other value is assigned.
PPP_DEFAULT_MP_MRRU	(UINT16) Maximum receive reconstructed unit size of an MP interface.

**Table 19-5. NU\_Set\_PPP\_Link\_Option Options (cont.)**

Option	Description
PPP_ENABLE_PFC	(UINT8) Enable or disable the use of Protocol Field Compression in PPP Headers for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_ACC	(UINT8) Enable or disable the use of Address Control Compression in PPP Headers for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_MAGIC_NUMBER	(UINT8) Enable or disable the use of Magic Number in PPP negotiations for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_ACCM	(UINT8) Enable or disable the use of Asynchronous Control Character Map in PPP negotiations for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_MRU	(UINT8) Enable or disable the use of Maximum Receive Unit for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_MP_MRRU	(UINT8) Enable or disable the use of maximum receive reconstructed unit size of an MP interface for this link. Possible values for this option are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_128BIT_ENCRYPTION	(UINT8) Whether 128-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_56BIT_ENCRYPTION	(UINT8) Whether 56-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>
PPP_ENABLE_40BIT_ENCRYPTION	(UINT8) Whether 40-bit MPPE encryption is enabled on this link. Possible values for this field are: <ul style="list-style-type: none"><li>• NU_TRUE</li><li>• NU_FALSE</li></ul>

**Table 19-5. NU\_Set\_PPP\_Link\_Option Options (cont.)**

Option	Description
PPP_ENABLE_ENCRYPTION_REQUIRED	(UINT8) Whether CCP negotiation is initiated for this link. Possible values for this option are: <ul style="list-style-type: none"> <li>• NU_TRUE</li> <li>• NU_FALSE</li> </ul>

- **optval**  
A pointer to the location from where the option value is read.
- **optlen**  
The size in bytes of the location pointed to by optval.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The options were successfully assigned.

### Example

```

UINT8      optval = NU_TRUE;
STATUS     status;

/* Enable Protocol Field Compression on the PPP link. */
status = NU_Set_PPP_Link_Option("ppp0", PPP_ENABLE_ACC,
                                &optval, sizeof(UINT8));

if (status == NU_SUCCESS)
{
    /* Link options updated successfully. */
}

```

### Related Topics

[PPP Services](#)

## NU\_PPP\_Last\_Activity\_Time

Returns the number of seconds since the last IP activity through the PPP link. This routine can be used by the application to determine if a connection can be closed due to inactivity.

### Usage

```
INT32 NU_PPP_Last_Activity_Time (CHAR *link_name);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to query.

### Return Values

- 0+  
The number of seconds since the last IP activity.
- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.

### Example

```
INT32      last_active;

/* Get last activity time for PPP link. */
last_active = NU_PPP_Last_Activity_Time "ppp0");

/* If the link has been idle for the last 10 minutes. */
if(last_active >= 600)
{
    /* Terminate the connection. */
    NU_PPP_Hangup(NU_FORCE, "ppp0");
}
```

### Related Topics

[PPP Services](#)



## NU\_PPP\_Negotiation\_Timeout

This service sets or gets the negotiation timeout for the link, in seconds.

### Usage

```
STATUS NU_PPP_Negotiation_Timeout (CHAR    *link_name,  
                                   INT      set,  
                                   UINT16  *neg_to_secs);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to set.
- **set**  
Set to NU\_TRUE to set the timeout, any other value gets the current timeout.
- **neg\_to\_secs**  
Pointer to the new negotiation timeout seconds or where to store the current seconds.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_INVALID\_PARM**  
There is a problem with the neg\_to\_secs parameter (it points to null).
- **NU\_SUCCESS**  
Successful completion of the service.

### Example

```
STATUS      status;  
INT32      timeout;  
  
/* Get current negotiation timeout for the PPP link. */  
status = NU_PPP_Negotiation_Timeout("ppp0", NU_FALSE, &timeout);  
  
if(status == NU_SUCCESS)  
{  
    /* 'timeout' contains the negotiation timeout  
     * interval in seconds.  
     */  
}
```

### Related Topics

[PPP Services](#)

## NU\_Obtain\_PPP\_Connection\_Status

This service checks the status of PPP connection.

### Usage

```
STATUS NU_Obtain_PPP_Connection_Status (CHAR *link_name);
```

### Arguments

- **link\_name**  
Pointer to the name of the waiting PPP link to abort.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_PPP\_DIALING**  
Attempting to connect.
- **NU\_PPP\_WAITING**  
Waiting for a client to dial-in.
- **NU\_PPP\_LINK\_NEGOTIATION**  
Modem connected, LCP negotiating.
- **NU\_PPP\_AUTHENTICATION**  
LCP open, authenticating.
- **NU\_PPP\_NETWORK\_NEGOTIATION**  
User authenticated, NCP negotiating.
- **NU\_PPP\_CONNECTED**  
NCP open, IP address obtained for the link.
- **NU\_PPP\_DISCONNECTED**  
LCP closed, modem disconnected.
- **NU\_PPP\_HANGING\_UP**  
LCP closed, modem disconnecting.
- **NU\_PPP\_WAITING\_ABORTED**  
No connection, server task no longer accepting dial-in clients.

### Example

```
STATUS          status;  
/* The loop below is used to wait until the "ppp0"  
 * link gets connected.  
 */  
do
```

```
{  
    /* Get current connection status. */  
    status = NU_Obtain_PPP_Connection_Status("ppp0");  
  
    /* Wait some time before polling again. */  
    NU_Sleep(TICKS_PER_SECOND);  
} while(status != NU_PPP_CONNECTED)
```

## Related Topics

[PPP Services](#)

## NU\_Abort\_Wait\_For\_PPP\_Client

This function allows a task to send an abort message so that the modem can be freed for other use.

### Usage

```
STATUS NU_Abort_Wait_For_PPP_Client (CHAR *link_name);
```

### Arguments

- **link\_name**  
Pointer to the name of the waiting PPP link to abort.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_BUSY**  
The link is currently connecting.
- **NU\_SUCCESS**  
The PPP link was successfully flagged to halt.

### Description

When an application task is acting as a PPP server waiting for a client to dial in, the modem is “reserved” by that task and cannot be used by another task during times when it is not busy. Once this function is called, it is the application’s responsibility to determine if the modem has been freed, since serial communication is inherently slow.

---

#### Note



If a connection is made just as this function is called, the connection will be dropped.

---

### Example

```
STATUS          status;

/* If a request has been made to abort waiting for
 * PPP connections. 'wait_abort_request' is a global
 * variable set to NU_TRUE when an abort is required.
 */
if(wait_abort_request == NU_TRUE)
{
    /* Abort waiting for clients. */
    NU_Abort_Wait_For_PPP_Client("ppp0");

    /* Reset the abort request. */
    wait_abort_request = NU_FALSE;
}
```

## Related Topics

[PPP Services](#)

## NU\_PPP\_Abort

This function aborts the phase when a PPP server is waiting for a client to dial in, or when a PPP client is in the dial phase or when a PPP negotiation phase is in progress.

### Usage

```
STATUS NU_PPP_Abort (CHAR *link_name);
```

### Arguments

- `link_name`  
Pointer to the name of the PPP link to abort.

### Return Values

- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.
- `NU_INVALID_MODE`  
Link is not in wait or negotiation state. Either it has been established or it has already been disconnected.
- `NU_SUCCESS`  
The PPP link was successfully aborted.

### Description

This function allows a task to send an abort message, prior to PPP connection establishment, so that the modem can be freed for other use. Once this function is called, it is the application's responsibility to determine if the modem has been freed, since serial communication is inherently slow.

---

#### Note



If a connection is made just as this function is called, the connection will be dropped.

---

### Example

```
STATUS      status;

/* If a request has been made to abort PPP link,
 * regardless of its current state.
 *
 * 'abort_request' is a global variable set to
 * NU_TRUE when an abort is required.
 */
if (abort_request == NU_TRUE)
{
    /* Abort the server wait or client dial phase. */
    status = NU_PPP_Abort("ppp0");

    /* If abort call failed because link is already
```

```
    * in a connected state and not in a server wait
    * or client dial phase
    */
    if (status != NU_SUCCESS)
    {
        /* Forcefully terminate the PPP link. Do not
        * event wait to reset the modem.
        */
        status = NU_PPP_Abort_Connection("ppp0");

        if(status == NU_SUCCESS)
        {
            /* Reinitialize the modem. */
            ...
        }
    }

    /* Reset the abort request. */
    abort_request = NU_FALSE;
}
```

## Related Topics

[PPP Services](#)

## NU\_PPP\_Abort\_Connection

This function aborts a PPP link when it is in the connected state. The remote node is not informed of the disconnection and all layers are brought down immediately.

---

### Note



This is a non-standard function and must only be called in situations that require an immediate connection termination.

---

---

### Note



The caller is responsible for hanging up and resetting the modem.

---

## Usage

```
STATUS NU_PPP_Abort_Connection (CHAR *link_name);
```

## Arguments

- **link\_name**  
Pointer to the name of the PPP link to abort.

## Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_INVALID\_MODE**  
Link is not in the connected state.
- **NU\_SUCCESS**  
The PPP link was successfully aborted.

## Example

```
STATUS      status;

/* If a request has been made to abort PPP link,
 * regardless of its current state
 *
 * 'abort_request' is a global variable set to
 * NU_TRUE when an abort is required.
 */
if(abort_request == NU_TRUE)
{
    /* Abort the server wait or client dial phase. */
    status = NU_PPP_Abort("ppp0");

    /* If abort call failed because link is already
     * in a connected state and not in a server wait
     * or client dial phase
     */
}
```



```
if (status != NU_SUCCESS)
{
    /* Forcefully terminate the PPP link. Do not
     * event wait to reset the modem.
     */
    status = NU_PPP_Abort_Connection("ppp0");

    if(status == NU_SUCCESS)
    {
        /* Reinitialize the modem. */
        ...
    }
}
/* Reset the abort request. */
abort_request = NU_FALSE;
}
```

## Related Topics

[PPP Services](#)

## PPP Metadata Options

This section describes the current PPP compile-time options, located in the *os/drivers/ppp/.metadata* file. In the option descriptions, “default” denotes the default setting. The options shipped by default are the default values specified in RFC 1661, where applicable, and should allow connection to any service provider.

Metadata compile-time options specific to HDLC configuration are defined in [HDLC Metadata Options](#). As well, metadata compile-time options for [DC Device Initialization](#) and [PPPoE Device Initialization](#) are detailed in their respective sections.

Additional [PPP Configuration Options](#) and [HDLC Configuration Options](#) can be found in the file *include/drivers/ppp\_cfg.h*.

---

### Note



All default values listed are only used during PPP initialization of the link. After initialization, these values can be changed by use of the PPP service [NU\\_Set\\_PPP\\_Link\\_Options](#).

---

```
option("use_chap") {
    default      true
    description "Enable support for CHAP authentication for client or
server. If connecting as a server, CHAP is the default
authentication protocol if both use_chap_ms1 and use_chap_ms2 are
defined false, unless changed by the application via
NU_Set_PPP_Link_Options."
}

option("use_pap") {
```

```
        default      true
        description "Enable support for PAP authentication for client or
server. If defined as false, a client will NAK the protocol in favor
of CHAP, MSCHAPv1, or MSCHAPv2, if defined."
    }

    option("use_chap_ms1") {
        default      true
        description "Enable support for MSCHAPv1 authentication for client
or server. If defined as false, clients will NAK the protocol in
favor of CHAP or MSCHAPv2, if defined. If connecting as a server,
MSCHAPv1 is the default authentication protocol, if MSCHAPv2 is
false unless changed by the application via NU_Set_PPP_Link_Options.
use_chap must also be true in order to allow MSCHAPv1 as a valid
authentication protocol."
    }

    option("use_chap_ms2") {
        default      true
        description "Enable support for MSCHAPv2 authentication for client
or server. If defined as false, clients will NAK the protocol in
favor of CHAP or MSCHAPv1, if defined. If connecting as a server,
MSCHAPv2 is the default authentication protocol unless changed by
the application via NU_Set_PPP_Link_Options. use_chap must also be
true in order to allow MSCHAPv2 as a valid authentication protocol."
    }

    option("enable_mppe") {
        default      false
        description "Enables the use of Microsoft Point-to-Point Encryption
Protocol (MPPE) over the PPP connections."
    }
}
```

---

#### Note



Nucleus PPP does not support the use of MPPE over Multilink connections.

---

```
    option("require_encryption") {
        default      true
        description "Defines whether MPPE is used over PPP connections by
default. When set to false, the use of MPPE encryption over PPP
connections becomes optional."
    }

    option("use_128_bit_encryption") {
        default      true
        description "Enable the use of 128-bit MPPE encryption on PPP
connections."
    }

    option("use_56_bit_encryption") {
        default      true
        description "Enable the use of 56-bit MPPE encryption on PPP
connections."
    }

    option("use_40_bit_encryption") {
```

```
    default      true
    description "Enable the use of 40-bit MPPE encryption on PPP
connections."
}

option("use_dns1") {
    default      true
    description "Retrieve a primary DNS server when establishing a new
PPP connection with a remote server."
}

option("use_dns2") {
    default      true
    description "Retrieve a secondary DNS server when establishing a new
PPP connection with a remote server."
}
```

## PPP Configuration Options

There are a number of configuration options available for PPP. All of these can be found in the file *include/drivers/ppp\_cfg.h*. The options shipped by default are the default values specified in RFC 1661, where applicable, and should allow connection to any service provider. They are made available for user configuration. Additional compile-time options are available in the [PPP Metadata Options](#).

These options are typically prefixed with the name of the module in which they are associated. The PPP configuration options described in the following sections exist in the LCP, NCP, and PPP modules, and are prefixed accordingly. Other options in *include/drivers/ppp\_cfg.h* may be prefixed with the module names for their particular device, such as HDLC and PPE (PPPoE). These are described in their respective chapters.

## INCLUDE\_PPP\_MP

Defining this macro to NU\_TRUE enables PPP Multilink Protocol. Refer to the [Multilink Protocol](#) section for more details on the protocol.

---

### Note



Nucleus PPP does not support the use of Microsoft Point-to-Point Encryption over Multilink connections.

---

## PPP\_MP\_DEFAULT\_MRRU\_SIZE

This specifies the default Maximum Receive Reconstructed Unit Size, when Multilink Protocol is enabled. MRRU specifies the maximum number of octets in the Information Fields of the reassembled packets. The size can be changed by using the [NU\\_Set\\_PPP\\_Link\\_Options](#) function. The default value is set at the minimum, 1500 octets. The maximum value can be 65536 octets.

## **PPP\_MP\_DEFAULT\_USE\_SHORT\_SEQ\_NUM**

While using Multilink Protocol, it defines, by default, whether short sequence number header format is used or not. The option advises the peer that the implementation wishes to receive fragments with short 12 bit sequence numbers rather than full 24 bit sequence numbers. This option can be changed by using the [NU\\_Set\\_PPP\\_Link\\_Options](#) function. The default is set to `NU_FALSE`.

## **INCLUDE\_PPP\_DC\_PROTOCOL**

Define this macro to `NU_TRUE` to use Direct Connect Cable Protocol. This involves direct cable (for example, Null Modem cable) between PPP client and PPP server.

## **PPP\_ENABLE\_CLI**

Define this macro to `NU_FALSE` if Caller Line Identification (CLI) is not supported by the modems/phone line, or if CLI support is not required.

## **PPP\_MAX\_ID\_LENGTH and PPP\_MAX\_PW\_LENGTH**

These specify the maximum number of characters of an ID and password. This applies to the ID and password used when dialing out and when a client is dialing in.

## **LCP\_MAX\_ECHO**

LCP echo request packets are used by PPP to determine if the link has been lost. At a time interval specified by `LCP_ECHO_VALUE`, PPP sends an echo request and expects to receive an echo reply. This macro is the number of echo request packets that are sent without a reply before the link is considered to be lost. If this constant is defined to be zero, then echo packets are not sent.

## **LCP\_ECHO\_VALUE**

This is the time between echo request packets. It is in clock ticks and is relative to the clock speed of the system. The time between requests should be long enough to allow for the reply to return.

## **LCP\_MAX\_CONFIGURE**

This is the maximum number of times to retry configuration of the link. This applies to both LCP and NCP negotiations.

## **LCP\_MAX\_TERMINATE**

This is the maximum number of times to retry terminating the link.

## **LCP\_MAX\_AUTHENTICATE**

This is the maximum number of times to retry authentication.

## **LCP\_TIMEOUT\_VALUE**

This is the time between retransmission of negotiation and authentication packets. It is in clock ticks and is relative to the speed of the hardware.

## **PPP\_TASK\_STACK\_SIZE**

This macro specifies the stack size of the PPP event handler task. It must be at least 2000 bytes.

## **PPP\_TASK\_PRIORITY**

This macro specifies the priority of the PPP event handler task. The default value is 3.

## **PPP\_TASK\_TIME\_SLICE**

This macro specifies the time slice of the PPP event handler task. Set to 0 to disable time slicing.

## **PPP\_TASK\_PREEMPT**

This macro specifies whether preemption is to be used for the PPP event handler task. Set this macro to `NU_PREEMPT` to enable preemption. Set it to `NU_NO_PREEMPT` to disable preemption.

## **DEBUG\_PKT\_TRACE**

Enables the capturing of all packets sent and received. By default this macro is commented. If debugging is required, then uncomment this macro.

## **PKT\_TRACE\_SIZE**

Defines the size of the packet trace buffer. When the buffer is filled, it wraps to the beginning and continue to log all packets. By default this macro is commented. If debugging is required, then uncomment this macro.

## **NU\_DEBUG\_PPP**

This macro enables printing of negotiating information. By default this macro is undefined. If debugging is required, then define this macro.

## PPP\_Printf

Defines where the printing of debug information is sent. This would normally be the standard `printf` as defined in *stdio.h*. However, some embedded systems tools do not define `printf`, so this constant can be defined as any function of your choice that takes a single string argument.

Each of the following constants turns printing on or off for debug information pertaining to those individual protocols. Check the top of each file for additional detailed configuration of the PPP output. This allows you to print only what you need in order to understand the flow of PPP link negotiation.

- `HDLC_DEBUG_PRINT_OK`
- `MDM_DEBUG_PRINT_OK`
- `PPP_DEBUG_PRINT_OK`
- `LCP_DEBUG_PRINT_OK`
- `CCP_DEBUG_PRINT_OK`
- `NCP_DEBUG_PRINT_OK`
- `CHAP_DEBUG_PRINT_OK`
- `MP_DEBUG_PRINT_OK`
- `PAP_DEBUG_PRINT_OK`

If `PPP_Printf` is defined as a standard `printf` or `sprintf` function, then the raw PPP negotiation packets can be output by defining this macro to `NU_TRUE`. Each byte in the packet is written as a hexadecimal digit, separated by spaces. There are 16 bytes per line of output, and each packet is preceded by its direction, 'in' or 'out'. This can be useful when negotiation fails and you are unable to obtain a log from the server, or to simply track the progression of a link negotiation.

## PPP\_LOG\_TO\_FILE

If your target has a file system or you are developing on MNT, then define this constant to write the above packet trace to the file defined by `PPP_LOG_FILENAME`. This is especially useful if you have a Technical Support issue or need to communicate the negotiation progression with others.

## PPP\_LOG\_FILENAME

This defines the location of the above log file `/drivers/ppp/src/ppp.c` that contains the code that is affected by this constant.

## Link Negotiations

PPP has been designed to negotiate all configuration options automatically without your help. Sometimes this negotiation fails for seemingly no reason. PPP does not guarantee that a successful connection will be made every time a connection is attempted. If connection fails, try to make the connection again. If, after several attempts, a connection is still not made, there may be a configuration option that the service provider does not understand, which is causing the negotiation to fail. Contact with the service provider may be necessary to match the configuration options specified in *include/drivers/ppp\_cfg.h* with those specified by the SP.

## Using PPP with SNMP

SNMP can be used to relay information about a PPP link according to the specifications in RFCs 1471, 1472, and 1473. These RFCs contain SNMP MIBs that are used to manage PPP link-specific data for LCP, Security (authentication), and NCP respectively. Like Nucleus NET, the collection of MIB statistics is completely independent of the SNMP agent, so the Nucleus SNMP agent module is not required to collect and use the PPP MIB information locally. Each MIB can be turned on or off by setting each of the following to `NU_TRUE` or `NU_FALSE` in *include/drivers/ppp\_cfg.h*.

- `INCLUDE_LCP_MIB`
- `INCLUDE_SEC_MIB`
- `INCLUDE_NCP_MIB`

### INCLUDE\_LCP\_MIB

This MIB contains information about a PPP link for a specific device, including the state of the LCP options that were negotiated. LCP MIB data is undefined until a PPP link is established and the state of LCP is open. See RFC 1471 for a complete specification of the PPP LCP MIB.

The MIB access functions for the LCP variables are located in */drivers/ppp/src/pml.c*.

### INCLUDE\_SEC\_MIB

The PPP security MIB contains information about the authentication protocols to be used for each link, whether it is acting as a server or a client, and also manages a database of user information that can be administered from a remote site. See RFC 1472 for complete specification of the PPP Security MIB.

The MIB access functions for the security configuration tables and structures are located in */drivers/ppp/src/pmsc.c*, while the functions for security secrets information and administration are located in */drivers/ppp/src/pmss.c*.

## INCLUDE\_NCP\_MIB

This MIB contains information about the IPCP network layer of a PPP link for a specific device, including the IP addresses that were negotiated. See RFC 1473 for a complete specification of the PPP NCP MIB.

The MIB access functions for the NCP variables are located in */drivers/ppp/src/pmn.c*.

If the optional Nucleus SNMP agent module is used to communicate these MIBs to a remote manager, then at least one of the previous MIBs must be included in the PPP library. The agent calls the MIB access functions in the files mentioned to collect the data, so these functions must be made available by including the appropriate MIB.

Note that a PPP device also updates statistics for the standard interface variables in MIB2 (RFC 1213), which is configured in *include/networking/net\_cfg.h*. MIB2 does not need to be enabled to use the PPP MIBs, and vice versa. See the [NET](#) chapter for more details on MIB2.

## IPv6 Over PPP

Nucleus NET versions starting at 5.1 include optional IPv6 capability, and this capability is extended to PPP by the addition of a new Network Control Protocol, namely IPV6CP, as defined in RFC 2472. For any PPP device that wishes to communicate IPv6 packets, this new NCP needs to be negotiated with the peer, so that a valid link-local (and optionally, a globally unique) IPv6 address can be established.

To use a specific PPP device to communicate IPv6 payloads, there are three things that must be ensured:

- IPv6 must be included in the Nucleus NET library. This is done by setting `INCLUDE_IPV6` in *include/networking/net\_cfg.h*. This allows IPv6 communications over any interface, in addition to the standard IPv4.
- Like in Nucleus NET, IPv6 payloads can be delivered without IPv4 payloads, as IPv4 can be removed from the system. IPv4 can be excluded in the Nucleus NET library. This is done by disabling `INCLUDE_IPV4` in *include/networking/net\_cfg.h*. If `INCLUDE_IPV4` is enabled and IPv4 negotiation is still not required, then it could be achieved by setting the flag in `NU_PPP_OPTIONS`. Refer to the [NU\\_Wait\\_For\\_PPP\\_Client](#) and [NU\\_Dial\\_PPP\\_Server](#) APIs for more details on setting the flag.
- Once the library modules are configured for IPv6, each individual PPP device can be configured to communicate IPv6 packets by simply adding the IPv6 flag to the group of device flags that are set when the device is installed. An application note describing an example of device configuration in greater detail can be found on SupportNet: <http://supportnet.mentor.com>



By following these steps, you can see that each device, of any type, can be configured individually to use (or not use) IPv6. See the [NET](#) chapter for more details on enabling and using IPv6.

Again, IPv4 communication can be excluded from the NET stack. This means that for any PPP link that negotiates the IPV6CP NCP for IPv6 addresses, the standard IPCP NCP is not negotiated if IPv4 communication is disabled.

## MPPE Over PPP

MPPE is used to secure communication over Point-to-Point connections. MPPE uses the RC4/ARC4 encryption algorithm. RC4 algorithm is not patented, but it is a trademark of RSA Data Security, Inc. The MPPE standard is defined by the following two RFCs:

- RFC 3078 - Microsoft Point to Point Encryption Protocol
- RFC 3079 - Deriving keys for use with Microsoft Point to Point Encryption Protocol

The encryption key provided to RC4 algorithm in MPPE is known as the session key. The session keys are derived from the MS-CHAPv1 or MS-CHAPv2 credentials. The length of the session key that is used to initialize the encryption table is negotiated during the PPP connection establishment. MPPE currently supports 40-bit, 56-bit, and 128-bit session keys. MPPE session keys are exchanged frequently. The exact frequency depends upon the type of key exchange mechanism negotiated. There are two types of key exchange mechanisms. They are described as follows:

- **Stateless Mode Key Changes:** In stateless encryption mode, the session key changes on every packet. In stateless mode, the sender **MUST** change its key before encrypting and transmitting each packet and the receiver **MUST** change its key after receiving, but before decrypting, each packet.
- **Stateful Mode Key Changes:** In stateful encryption mode, the sender **MUST** change its key before encrypting and transmitting a flag packet and the receiver **MUST** change its key after receiving, but before decrypting, a "flag" packet. Any encrypted packet can be marked as a flag packet. The flag packet is made according to steps defined in RFC 3078.

The type of key exchange mechanism and the strength of encryption are determined during PPP connection establishment. MPPE uses Compression Control Protocol (CCP) to negotiate the MPPE specific options with the peer.

## Control Compression Protocol Over PPP

The Control Compression Protocol, as the name implies, provides a control mechanism to exchange configuration options for any compression algorithm to be used on PPP connections. However, the Control Compression Protocol is also the control protocol for encryption

algorithm namely, MPPE. The configuration options of MPPE are negotiated using PPP Compression Control Protocol.

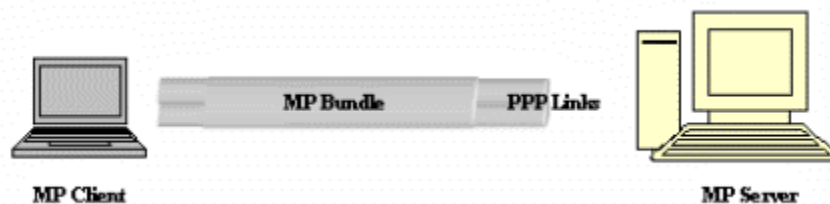
The Compression Control Protocol negotiations do not start until Link Control Protocol has exchanged the link options and authentication is complete. As soon as authentication is complete, the Compression Control Protocol starts negotiating the MPPE configuration options.

The Compression Control Protocol is identical to Link Control Protocol. RFC 1962 defines the Control Compression Protocol standard.

## Multilink Protocol

Multilink Protocol (MP) provides greater bandwidth by providing multiple independent links between two network nodes. The multiple links form one virtual link (called a bundle), which is the only link visible to the higher layer protocols. This link is used to send and receive data. When transmitting, packets are fragmented and sent over all physical PPP links that exist between the two PPP peers. On reception, these fragments are reassembled into the original packet, which is dispatched to the higher layer protocols. The following figure shows two peers communicating with each other on multiple links that form a bundle.

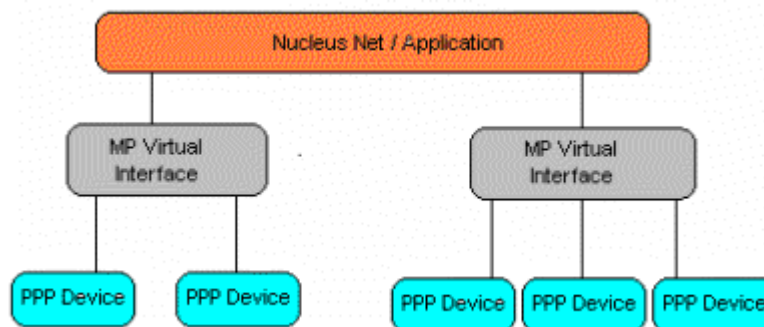
**Figure 19-2. A Multilink PPP Connection**



MP provides a method for splitting, recombining, and sequencing datagrams across multiple logical data links. This approach reduces latency and increases the effective maximum receive unit (MRU). Multilink operation coordinates multiple independent links between a fixed pair of systems, which provides a virtual link between those systems with greater bandwidth than any of the constituent logical data links.

The following figure illustrates the relationship between the Nucleus MP virtual interface, Nucleus PPP, Nucleus NET, and user applications. There can be multiple MP virtual interfaces and each virtual interface can have multiple PPP devices.

**Figure 19-3. Relationship Between Components Involved in a Multilink Connection**



Multilink is part of Nucleus PPP and can be included/excluded from the build at compile time. Now developers can make use of Nucleus NET's IPv4 and IPv6 stacks in their point-to-point communication, with the added advantage of higher speed that is the characteristic of Multilink.

Nucleus MP can also be used in conjunction with Nucleus L2TP to provide faster multilink VPN connections.

The `/drivers/ppp/src/ppp_mpfrag.c` file organizes incoming fragments and fragments outgoing packets that are to be transmitted on links associated with an MP virtual interface. Many of the routines that provide the handling of Multilink Protocol are defined in `/drivers/ppp/src/ppp_mp.c`.

The user level services are listed in the following [MP Services](#) section .

## MP Services

This section lists the available MP services and provides a brief description of each.

- [NU\\_MP\\_Get\\_Virt\\_If\\_By\\_Device](#)
- [NU\\_MP\\_Get\\_Virt\\_If\\_By\\_User](#)
- [NU\\_MP\\_Terminate\\_Links](#)
- [NU\\_MP\\_Get\\_Opt](#)

## NU\_MP\_Get\_Virt\_If\_By\_Device

This function returns the index of the MP virtual interface associated with a PPP link (device).

### Usage

```
STATUS NU_MP_Get_Virt_If_By_Device (UINT32 *mp_ifindex,  
                                   CHAR    *link_name);
```

### Arguments

- **mp\_ifindex**  
Interface index of the MP virtual device which is associated with the PPP link name passed in.
- **link\_name**  
Pointer to the PPP link name.

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_NOT\_PRESENT**  
MP device not found.

### Example

```
STATUS  status;  
UINT32  mp_ifindex;  
  
...  
  
/* Get the MP virtual interface associated with the  
 * device with the interface name, "ppp0".  
 */  
  
status = NU_MP_Get_Virt_If_By_Device(&mp_ifindex, "ppp0");
```

### Related Topics

[MP Services](#)

## NU\_MP\_Get\_Virt\_If\_By\_User

This function returns the MP virtual interface index that matches the MP user name.

### Usage

```
STATUS NU_MP_Get_Virt_If_By_User (UINT32 *mp_ifindex,  
                                CHAR    *user_name);
```

### Arguments

- **mp\_ifindex**  
Interface index of the MP virtual device which is associated with the PPP user name
- **user\_name**  
Pointer to the user name of the MP interface.

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_NOT\_PRESENT**  
MP device not found.

### Example

```
STATUS status;  
UINT32 mp_if_index;  
  
/* Get the index of the MP interface associated with  
 * the user name, "my_user_name".  
 */  
status = NU_MP_Get_Virt_If_By_User(&mp_ifindex, "my_user_name");
```

### Related Topics

[MP Services](#)

## NU\_MP\_Terminate\_Links

This function hangs up all links associated with an MP virtual interface.

### Usage

```
STATUS NU_MP_Terminate_Links (UINT32 mp_ifindex);
```

### Arguments

- **mp\_ifindex**  
Interface index of the MP virtual device.

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_NOT\_PRESENT**  
MP virtual interface not found or a PPP link could not be terminated.

### Example

```
STATUS  status;  
UINT32  if_index;  
  
/* Get device index associated with MP virtual interface. */  
if_index = NU_IF_NameToIndex("mp_example");  
  
/* Terminate all links associated with this MP virtual  
 * interface.  
 */  
status = NU_MP_Terminate_Links(if_index);
```

### Related Topics

[MP Services](#)

## NU\_MP\_Get\_Opt

This function gets the options associated with the MP virtual interface.

### Usage

```
STATUS NU_MP_Get_Opt (UINT32 mp_ifindex,  
                     INT     optname,  
                     VOID    *optval,  
                     INT     *optlen)
```

### Arguments

- **mp\_ifindex**  
Interface index of the MP virtual interface.
- **optname**  
Name of the option to get. Refer to [Table 19-6](#) and [Table 19-7](#) for valid options for Discriminator Class and PPP.

**Table 19-6. NU\_MP\_Get\_Opt Discriminator Class Options**

Discriminator Class	Description
MP_CLASS_NULL	This class is the default value if no endpoint discriminator is specified.
MP_CLASS_IP	An address in this class contains an IP address. Max length is four.
MP_CLASS_MAC	An address in this class contains an IEEE 802.1 MAC address in canonical format. Max length is six.
MP_CLASS_PSND	Public Switched Network Directory Number. An address in this class contains an octet sequence as defined by I.331 representing an international telephone directory number suitable for use to access the endpoint via the public switched telephone network. Max length is 15.

**Table 19-7. NU\_MP\_Get\_Opt PPP Options**

Option	Description
PPP_MP_USER_NAME	It returns the MP user name.
PPP_MP_END_DISCRIMINATOR	This returns the Endpoint discriminator of the peer. This option represents identification of the peer system.
PPP_MP_END_DISCRIMINATOR_CLASS	This returns the class of the Endpoint discriminator. Refer to the discriminator class <a href="#">Table 19-6</a> for the valid options.



Table 19-7. NU\_MP\_Get\_Opt PPP Options (cont.)

Option	Description
PPP_MP_MRRU	This returns the MRRU for the MP interface. MRRU (Maximum Receive Reconstructed Unit) specifies the maximum number of octets in the Information Fields of the reassembled packets.
PPP_MP_GET_ALL_LINKS	This option returns a string that contains names of all the links associated with the passed MP interface. The names are delimited by a comma (.).
PPP_MP_IS_INTERFACE	This returns NU_SUCCESS if the passed index is an MP interface.
PPP_MP_NEXT_INTERFACE	This returns the interface index of the next MP interface.

- **optval**  
Pointer to the location where the option value can be written.
- **optlen**  
Pointer to the size in bytes of the location pointed to by optval.

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_INVALID\_OPTION**  
The option does not exist.
- **NU\_INVALID\_PARM**  
Either optval or optlen parameter has problems.
- **NU\_NOT\_PRESENT**  
MP interface passed could not be found.

### Example

```
UINT32  mp_ifindex;
UINT8   peer_disc_value[15];
UINT8   peer_disc_class;
UINT8   len_disc = 15;
UINT8   len_disc_class = 1;

/* Get the interface index of the virtual interface with
 * the name "mp_example".
 */
mp_ifindex = NU_IF_NameToIndex("mp_example");

/* Get the endpoint discriminator of the peer. */
NU_MP_GET_OPT(mp_ifindex, PPP_MP_END_DISCRIMINATOR,
```

```
        &peer_disc_value, &len_disc);

/* Get the discriminator class of the value got above. */
NU_MP_GET_OPT(mp_ifindex, PPP_MP_END_DISCRIMINATOR_CLASS,
              &peer_disc_class, &len_disc_class);
```

## Related Topics

[MP Services](#)

## MP Initialization

Multilink Protocol is part of Nucleus PPP and needs to be enabled by setting the macro to `NU_TRUE` in `include/drivers/ppp_cfg.h`. The following examples assume that Nucleus NET has been initialized and that there are two PPP devices (UARTs) for dialing through the modems (HDLC) that are attached to these devices.

### Example: MP Client

The following example demonstrates how a MP client makes a connection to a server using two devices (HDLC interfaces). First, two PPP devices are initialized and, then, they dialed the PPP MP Server on a specified number, one by one. Note that the MP virtual device must be initialized before making the MP call on the devices.

```
NU_DEVICE      device[3];
NU_PPP_OPTIONS ppp_options;

/* Initialize the first PPP device, for making a call. */
device[0].dv_name = "ppp0";
device[0].dv_hw.uart.com_port    = DEFAULT_UART_PORT0;
device[0].dv_hw.uart.baud_rate   = DEFAULT_UART_BAUD;
device[0].dv_hw.uart.parity      = DEFAULT_UART_PARITY;
device[0].dv_hw.uart.stop_bits   = DEFAULT_UART_STOP;
device[0].dv_hw.uart.data_bits   = DEFAULT_UART_DATA;
device[0].dv_hw.uart.data_mode   = DEFAULT_UART_MODE;
device[0].dv_init = HDLC_Initialize;
device[0].dv_flags = (DV_POINTTOPOINT | DV_NOARP);

/* Initialize the second PPP device, for making a call. */
device[1].dv_name = "ppp1";
device[1].dv_hw.uart.com_port    = DEFAULT_UART_PORT1;
device[1].dv_hw.uart.baud_rate   = DEFAULT_UART_BAUD;
device[1].dv_hw.uart.parity      = DEFAULT_UART_PARITY;
device[1].dv_hw.uart.stop_bits   = DEFAULT_UART_STOP;
device[1].dv_hw.uart.data_bits   = DEFAULT_UART_DATA;
device[1].dv_hw.uart.data_mode   = DEFAULT_UART_MODE;
device[1].dv_init = HDLC_Initialize;
device[1].dv_flags = (DV_POINTTOPOINT | DV_NOARP);

/* Initialize MP virtual interface. */
device[2].dv_name = "my_example";
device[2].dv_init = PPP_MP_Init;
device[2].dv_flags = (DV_POINTTOPOINT | DV_VIRTUAL_DEV);

/* Register the devices. */
if(NU_Init_Devices(device, 3) != NU_SUCCESS)
{
    printf("Error at call to NU_Init_Devices().\n");
    DEMOI_Exit(__LINE__);
}

/* Set the user name and password for the MP interface. */
NU_Set_PPP_Login("fred", "fish", "my_example");
```

```
/* Clear the options structure. */
UTL_Zero(&ppp_options, sizeof(NU_PPP_OPTIONS));

/* Set the PPP options for dialing the MP server. */
ppp_options.ppp_num_to_dial = "1234";
ppp_options.ppp_link = "ppp0";
ppp_options.ppp_mp_link = "my_example";

/* Dial the PPP MP Server. */
status = NU_Dial_PPP_Server(&ppp_options);

/* Clear the options structure. */
UTL_Zero(&ppp_options, sizeof(NU_PPP_OPTIONS));

/* Set the PPP options for dialing the MP server on the
 * other link.
 */
ppp_options.ppp_num_to_dial = "1235";
ppp_options.ppp_link = "ppp1";
ppp_options.ppp_mp_link = "my_example";

/* Dial the PPP MP Server. */
status = NU_Dial_PPP_Server(&ppp_options);

...
```

## Example: MP Server

The following example demonstrates how a MP server accepts connections from an MP client using two devices (HDLC interfaces). The demo is almost the same as that of the PPP server demo. Note that an MP virtual device is not required to be created and initialized here, as it was required in the MP Client example. It will automatically be created and initialized whenever a new connection is established.

---

### Note



Links between two MP peers will always have the same IP address, if they are associated with the same MP virtual interface.

---

```
NU_DEVICE    device[2];
PPP_OPTIONS  ppp_options;
UINT8        local_ip[]    = {192, 168, 0, 138};
UINT8        remote_ip[]   = {192, 168, 0, 139};

/* Initialize first PPP device, for accepting a call. */
device[0].dv_name = "ppp0";
device[0].dv_hw.uart.com_port    = DEFAULT_UART_PORT0;
device[0].dv_hw.uart.baud_rate   = DEFAULT_UART_BAUD;
device[0].dv_hw.uart.parity      = DEFAULT_UART_PARITY;
device[0].dv_hw.uart.stop_bits   = DEFAULT_UART_STOP;
device[0].dv_hw.uart.data_bits   = DEFAULT_UART_DATA;
device[0].dv_hw.uart.data_mode   = DEFAULT_UART_MODE;
device[0].dv_init = HDLC_Initialize;
device[0].dv_flags = (DV_POINTTOPOINT | DV_NOARP);
```

```
/* Initialize second PPP device, for accepting a call. */
device[1].dv_name = "ppp1";
device[1].dv_hw.uart.com_port = DEFAULT_UART_PORT1;
device[1].dv_hw.uart.baud_rate = DEFAULT_UART_BAUD;
device[1].dv_hw.uart.parity = DEFAULT_UART_PARITY;
device[1].dv_hw.uart.stop_bits = DEFAULT_UART_STOP;
device[1].dv_hw.uart.data_bits = DEFAULT_UART_DATA;
device[1].dv_hw.uart.data_mode = DEFAULT_UART_MODE;
device[1].dv_init = HDLC_Initialize;
device[1].dv_flags = (DV_POINTTOPOINT | DV_NOARP);

/* Register the device. */
if (NU_Init_Devices(DEMOI_Devices, 2) != NU_SUCCESS)
{
    printf("Error at call to NU_Init_Devices().\n");
    DEMOI_Exit(__LINE__);
}

/* Clear out the structure. */
UTL_Zero(&ppp_options, sizeof(NU_PPP_OPTIONS));

/* Set the name of the link for which options are going to be
 * set.
 */
ppp_options.ppp_link = "ppp0";

/* Set the IPv4 address that will be assigned to the calling
 * PPP client.
 */
ppp_options.ppp_remote_ip4 = remote_ip;

/* Set the IPv4 address that will be used on this end of the
 * PPP link.
 */
ppp_options.ppp_local_ip4 = local_ip;

/* Set the IP address that will be assigned to a remote
 * clients.
 */
NU_Set_PPP_Client_IP_Address(&ppp_options);

/* Wait for a client to call on the device, "ppp0". */
status = NU_Wait_For_PPP_Client(&ppp_options);

/* Set the same PPP options for the second link. */
ppp_options.ppp_link = "ppp1";

/* Wait for a client to call on the device, "ppp1". */
status = NU_Wait_For_PPP_Client(&ppp_options);

...
```

## HDLC Interface


This section contains an [HDLC Introduction](#) and information about:

- [HDLC Services](#)
- [HDLC Device Initialization](#)
- [HDLC Metadata Options](#)
- [HDLC Configuration Options](#)

## HDLC Introduction

The HDLC interface is the default serial interface that ships with Nucleus PPP. It essentially encapsulates the serial routines that control an onboard UART (or other serial hardware device) and the modem control routines that communicate with the modem. On most targets, the default serial driver is used to handle the actual transmission of serial data. However, some ports of PPP still include their own module for driving the UART. The file */drivers/ppp/src/hdlc.c* organizes the incoming serial characters into Nucleus NET buffers that can be used by the upper layers, and vice versa for transmitted data. */drivers/ppp/src/mdm.c* contains all the modem control routines. Many of the routines are services that allow the application to specify how the modem is used. These services are listed in the [HDLC Services](#) section.

---

 **Note** Some portions of this chapter may not apply to your particular target (mainly those dealing with modem control functions, since some embedded UARTs do not contain modem control functionality).

---

## HDLC Services

All of the following services can be more accurately described as modem services. Most of these services are for driving the modem and take the name of the device as a parameter. This is the same name used in the device initialization structure for each device. By naming the devices and supplying the device name to each service, the ability to support multiple PPP links is obtained.

The following sections list the HDLC services and provide a brief description of each.

- [NU\\_Reset\\_Modem](#)
- [NU\\_Carrier](#)
- [NU\\_Change\\_Communication\\_Mode](#)
- [NU\\_Modem\\_Control\\_String](#)

- [NU\\_Modem\\_Rings\\_To\\_Answer](#)
- [NU\\_Terminal\\_Data\\_Ready](#)
- [NU\\_Get\\_Terminal\\_Char](#)
- [NU\\_Purge\\_Terminal\\_Buffer](#)
- [NU\\_Put\\_Terminal\\_Char](#)
- [NU\\_Modem\\_Get\\_Remote\\_Num](#)
- [NU\\_Modem\\_Set\\_Local\\_Num](#)

## NU\_Reset\_Modem

This function resets and initializes the modem before dialing out.

### Usage

```
VOID NU_Reset_Modem (CHAR *if_name);
```

### Arguments

- **if\_name**  
Pointer to the name of the interface.

### Return Values

- none

### Example

```
/* Reset and initialize the modem to prepare to dial out */  
NU_Reset_Modem(ppp0);  
  
/* Continue to set PPP_Options and dial the number. */  
...
```

### Related Topics

[NU\\_Modem\\_Get\\_Remote\\_Num](#)

[NU\\_Modem\\_Set\\_Local\\_Num](#)



## NU\_Carrier

This function detects the presence of a carrier.

---

### Note



This service is only valid on platforms for which the UART has the ability to detect the presence of a carrier.

---

## Usage

```
STATUS NU_Carrier (CHAR *link_name);
```

## Arguments

- link\_name

Pointer to the name of the PPP link to which this service should be applied.

## Return Values

- NU\_TRUE  
A carrier is present.
- NU\_FALSE  
A carrier is not present.
- NU\_INVALID\_LINK  
The link name supplied is not a valid PPP device.

## Example

```
STATUS status;

/* Detect presence of a carrier for the "ppp0" link. */
status = NU_Carrier("ppp0");

if(status == NU_TRUE)
{
    /* Carrier is present. */
    ...
}
else if(status == NU_FALSE)
{
    /* Carrier not present. */
    ...
}
else
{
    /* An error occurred. */
    ...
}
```

## Related Topics

[HDLC Services](#)

## NU\_Change\_Communication\_Mode

This service switches the PPP driver between network mode and terminal mode. In network mode, arriving characters are assumed to be part of a PPP packet and are routed to the PPP ISR. In terminal mode, received characters are routed to the modem ISR, which makes them available to the application layer by use of the “terminal” functions, for example `NU_Get_Terminal_Char`, `NU_Put_Terminal_Char`, and so on.

### Usage

```
STATUS NU_Change_Communication_Mode (INT    mode,  
                                     CHAR  *link_name);
```

### Arguments

- `mode`  
The mode of communication desired. The only two valid values are: `PPP_NETWORK_COMMUNICATION` and `PPP_TERMINAL_COMMUNICATION`.
- `link_name`  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- `NU_SUCCESS`  
Successful completion of the service.
- `NU_INVALID_MODE`  
The mode parameter is not valid.
- `NU_INVALID_LINK`  
The link name supplied is not a valid PPP device.

### Example

```
STATUS      status;  
  
/* Change communication mode to terminal mode. */  
status = NU_Change_Communication_Mode(PPP_NETWORK_COMMUNICATION, "ppp0");  
  
if(status == NU_SUCCESS)  
{  
    /* Mode successfully changed to terminal. */  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Modem\_Control\_String

This service is used to send a control string to the modem. By default, a delay of 100 ms is inserted between each character in order to conform to the modem receive logic. If a greater delay is desired, a '~' can be inserted into the control string. A delay of a half second occurs for each '~' in the control string.

### Usage

```
STATUS NU_Modem_Control_String (CHAR *string,  
                                CHAR *link_name);
```

### Arguments

- **string**  
Pointer to the ASCII string to be sent to the modem.
- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_INVALID\_POINTER**  
The string pointer is NULL.
- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.

### Example

```
STATUS  status;  
  
/* Reset the modem. */  
status = NU_Modem_Control_String("ATZ^M", "ppp0");  
  
if(status == NU_SUCCESS)  
{  
    /* Modem reset successfully. */  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Modem\_Rings\_To\_Answer

This service is used to change the number of rings on which the modem answers.

### Usage

```
STATUS NU_Modem_Rings_To_Answer (CHAR  *link_name,  
                                UINT8  num_rings);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.
- **num\_rings**  
Number of rings on which the modem answers (0-255).

### Return Values

- **NU\_SUCCESS**  
Successful completion of the service.
- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.

### Example

```
STATUS  status;  
  
/* Set the number of rings the modem answers on to 5. */  
status = NU_Modem_Rings_To_Answer("ppp0", 5);  
  
if(status == NU_SUCCESS)  
{  
    /* The modem now answers after 5 rings. */  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Terminal\_Data\_Ready

This service is used to check the terminal mode data buffer for data.

### Usage

```
STATUS NU_Terminal_Data_Ready (CHAR *link_name);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- **NU\_TRUE**  
Data is present.
- **NU\_FALSE**  
No data is present.
- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.

### Example

```
STATUS  status;  
CHAR    ch;  
  
/* Get the status of the terminal data buffer. */  
status = NU_Terminal_Data_Ready("ppp0");  
  
if(status == NU_TRUE)  
{  
    /* Loop until the terminal data buffer is empty. */  
    do  
    {  
        /* Retrieve and discard a character from the  
         * terminal buffer.  
         */  
        status = NU_Get_Terminal_Char(&ch, "ppp0");  
    } while(status == NU_SUCCESS);  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Get\_Terminal\_Char

This service retrieves a character from the receive buffer. This service should only be used when in terminal mode. In network mode, received characters are routed to the TCP/IP protocol stack.

### Usage

```
STATUS NU_Get_Terminal_Char (CHAR *c,  
                             CHAR *link_name);
```

### Arguments

- **c**  
Pointer to the location to store the received character.
- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- **NU\_SUCCESS**  
A character was returned.
- **NU\_NO\_DATA**  
No data was available.
- **NU\_INVALID\_POINTER**  
A pointer parameter points to NU\_NULL.
- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.

### Example

```
STATUS  status;  
CHAR    ch;  
  
/* Get the status of the terminal data buffer. */  
status = NU_Terminal_Data_Ready("ppp0");  
  
if(status == NU_TRUE)  
{  
    /* Loop until the terminal data buffer is empty. */  
    do  
    {  
        /* Retrieve and discard a character from the terminal buffer. */  
        status = NU_Get_Terminal_Char(&ch, "ppp0");  
    } while(status == NU_SUCCESS);  
}
```

## Related Topics

[HDLC Services](#)



## NU\_Purge\_Terminal\_Buffer

This service purges the terminal mode receive buffer of any characters.

### Usage

```
STATUS NU_Purge_Terminal_Buffer (CHAR *link_name);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The buffer was cleared.

### Example

```
STATUS  status;  
CHAR    ch;  
  
/* Get the status of the terminal data buffer. */  
status = NU_Terminal_Data_Ready("ppp0");  
  
/* If the buffer is not empty */  
if(status == NU_TRUE)  
{  
    /* Purge the terminal data buffer. */  
    status = NU_Purge_Terminal_Buffer("ppp0");  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Put\_Terminal\_Char

This service sends a character to the serial port. It is used to send characters when in terminal mode.

### Usage

```
STATUS NU_Put_Terminal_Char (CHAR c,  
                             CHAR *link_name);
```

### Arguments

- **c**  
The character to be sent to the serial port.
- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The character was sent.

### Example

```
STATUS status;  
  
/* Put a single character in the terminal buffer. */  
status = NU_Put_Terminal_Char('A', "ppp0");  
  
if(status == NU_SUCCESS)  
{  
    /* Character successfully put in terminal buffer. */  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Modem\_Get\_Remote\_Num

This service returns the telephone number of the remote peer. If the local peer is acting as a PPP client, then it is the number it has dialed. Also, if the local peer is acting as a PPP server, then it is the number of the dialing client.

---

### Note



Modems and phone lines must support caller line identification (CLI). If the CLI service is available, refer to the section [PPP Configuration Options](#) to turn this facility on.

---

### Usage

```
STATUS NU_Modem_Get_Remote_Num (CHAR *link_name,  
                                CHAR *remote_num);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.
- **remote\_num**  
Pointer to the phone number of the remote peer.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The successful completion of the service.

### Example

```
STATUS  status;  
CHAR    remote_num[30];  
  
/* Get the remote peer's number. */  
status = NU_Modem_Get_Remote_Num("ppp0", remote_num);  
  
if(status == NU_SUCCESS)  
{  
    /* 'remote_num' contains the remote peer's number. */  
}
```

### Related Topics

[HDLC Services](#)

## NU\_Modem\_Set\_Local\_Num

This service sets the number of the telephone line that is attached to the modem (link\_name).

### Usage

```
STATUS NU_Modem_Set_Local_Num (CHAR *link_name,  
                               CHAR *local_num);
```

### Arguments

- **link\_name**  
Pointer to the name of the PPP link to which this service should be applied.
- **local\_num**  
Pointer to the phone number of the local modem.

### Return Values

- **NU\_INVALID\_LINK**  
The link name supplied is not a valid PPP device.
- **NU\_SUCCESS**  
The successful completion of the service.

### Example

```
STATUS status;  
CHAR local_num[] = "1234567";  
  
/* Set the local number to 1234567. */  
status = NU_Set_Local_Num("ppp0", local_num);  
  
if(status == NU_SUCCESS)  
{  
    /* Local number set successfully. */  
}
```

### Related Topics

[HDLC Services](#)

## HDLC Device Initialization

As mentioned briefly in the [PPP Overview](#) chapter, one of the ways that PPP network applications differ from network applications that use Ethernet or some other network medium is in the device initialization. A Nucleus PPP device is initialized differently as compared to the other devices in the Nucleus system. It uses an already initialized device (UART for example) and so it is just a matter of providing the correct information to the Nucleus NET [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) service.

By default, if PPP is enabled in the system, one of the PPP interfaces gets initialized based on the configurations enabled in the PPP *.metadata* file. To enable an HDLC interface, please ensure that the following options are configured as follows:

```
option("enable_serial") {
    default true
    description "This enables PPP communication over the serial
                interface."
}

option("dc_protocol_enable") {
    default      false
    description "Enable the interface to use Direct Connect Cable Protocol"
}
```

## HDLC Metadata Options

This section describes the current HDLC compile-time options, located in the *os/drivers/ppp/.metadata* file. In the option descriptions, “default” denotes the default setting for the option. The compile-time options for HDLC are specific to the serial device and modem, and complement the options of Nucleus PPP.

The options shipped by default are the default values specified in RFC 1661, where applicable, and should allow connection to any service provider. Metadata compile-time options specific to PPP configuration are defined in [PPP Metadata Options](#). As well, metadata compile-time options for [DC Device Initialization](#) and [PPPoE Device Initialization](#) are detailed in their respective sections.

Additional [PPP Configuration Options](#) and [HDLC Configuration Options](#) can be found in the file *include/drivers/ppp\_cfg.h*.

---

### Note



All default values listed are only used during PPP initialization of the link. After initialization, these values can be changed by use of the PPP service [NU\\_Set\\_PPP\\_Link\\_Options](#).

---

```
option("use_accm") {
    default      true
    description "Enable support for asynchronous control character
                mapping in PPP."
}

option("use_mru") {
    default      false
    description "Defines whether the maximum receive unit is negotiated.
                This option should only be used if the MRU for the link has been
                changed from the default value of 1500 bytes."
}
```

```
option("use_pfc") {
    default      false
    description "Defines whether protocol field compression is used.
    Using this compression saves 1 byte per packet sent/received."
}

option("use_acc") {
    default      false
    description "Defines whether address and control field compression
    is used. Using this compression saves 2 bytes per packet
    sent/received."
}

option("use_magic") {
    default      true
    description "Defines whether the LCP magic number is negotiated.
    Since the LCP echo packets make use of the magic number, this option
    is recommended for all links."
}
```

## HDLC Configuration Options

The configuration options for HDLC are specific to the serial device and modem, and complement the configuration options of Nucleus PPP. The HDLC compile-time options are found in the *os/drivers/ppp/.metadata* file and are described in [HDLC Metadata Options](#). Additional configuration options are found in the file *include/drivers/ppp\_cfg.h* and are further described in this section.

The options shipped by default are the default values specified in RFC 1661, where applicable, and should allow connection to any service provider. They are made available for user configuration. All options and their valid settings are listed in the following sections. The use for some of these options may be confusing. See [Figure 19-4](#) and [Figure 19-5](#) at the end of this section for clarification on how to use [HDLC\\_MAX\\_HOLDING\\_PACKETS](#) and [HDLC\\_MAX\\_HOLDING\\_PACKETS\\_PTR](#), and [HDLC\\_MAX\\_TX\\_QUEUE\\_PTRS](#).

### HDLC\_FOREIGN\_DEFAULT\_ACCM and HDLC\_LOCAL\_DEFAULT\_ACCM

This is the default asynchronous control character map. This specifies which characters, when transmitted, should be made transparent to the hardware. This is a 32-bit map and each bit corresponds to the ASCII equivalent character. Since some hardware devices use control characters, this map provides a way to hide the control characters from the hardware. If it is known exactly what control characters may affect the specific hardware used, then this can be set to those characters. If it is not known, then the default value should be used to mask all ASCII characters below 32. There are default values for both the foreign and local ends of the PPP link. The local ACCM can be changed to fit any particular hardware being used. In most cases, if the hardware does not respond to control codes on the line, this value can be set to 0x0.

The foreign ACCM default value should be left as 0xffffffff. During LCP negotiation, the foreign host informs the local end of the link to what its ACCM should be set.

## MDM\_RINGS\_TO\_ANSWER\_ON

This define is used to put the modem into answer mode. It control sthe number of rings needed before the modem answers the phone. This option does not apply to PPPoE links.

## HDLC\_MAX\_HOLDING\_PACKETS and HDLC\_MAX\_HOLDING\_PACKETS\_PTR

HDLC\_MAX\_HOLDING\_PACKETS is used to declare an array that holds the packet that is currently being received. Since HDLC packets are received a byte at a time, there needs to be buffers in place to hold the packet until it has been completely received. These buffers are link.rx\_ring; a ring of buffers for each PPP link. In [Figure 19-4](#), it can be seen that once the packet has been fully received, it is passed to the RX HISR by means of a second array. This array is of size HDLC\_MAX\_HOLDING\_PACKETS\_PTR. It consists only of pointers that point to the packets in the link.rx\_ring. When the RX HISR is processing a packet, the data it is processing is still on a link's receive ring buffer. Therefore, this ring buffer needs to be large enough to hold already received packets and still have room to receive more packets until the RX HISR is done processing previously received ones. Since the RX HISR is responsible for processing packets for all UARTs, its pointer array, \_ppp\_rx\_queue, needs to be large enough to point to received packets from multiple UARTs, if more than one UART is being used. A general rule for the worst-case situation would be three HDLC\_MAX\_HOLDING\_PACKET per UART and the total of that for HDLC\_MAX\_HOLDING\_PACKETS\_PTR.

Since HDLC\_MAX\_HOLDING\_PACKETS\_PTR is only an array of pointers, this is not that expensive of a configuration. HDLC\_MAX\_HOLDING\_PACKET, on the other hand, is an array of PPP frames. Since a PPP frame can be no smaller than 1500 bytes, this option can consume large amounts of memory if there are a significant number of UARTs being used.

## HDLC\_MAX\_TX\_QUEUE\_PTRS

This macro is used to declare an array that holds pointers to PPP device structures. When the UART code has completed transmission of a PPP packet, it puts the address of the PPP device that has completed transmission in this queue. Then it activates the TX HISR, please see [Figure 19-5](#). The TX HISR pulls the device pointer out of this queue, and if there is another packet ready for transmission, the HISR starts its transmission. For a worst-case situation, the size of this macro should be large enough to hold two device pointers from each PPP device in the system.

**Figure 19-4. HDLC\_MAX\_HOLDING\_PACKETS and HDLC\_MAX\_HOLDING\_PACKETS\_PTR**

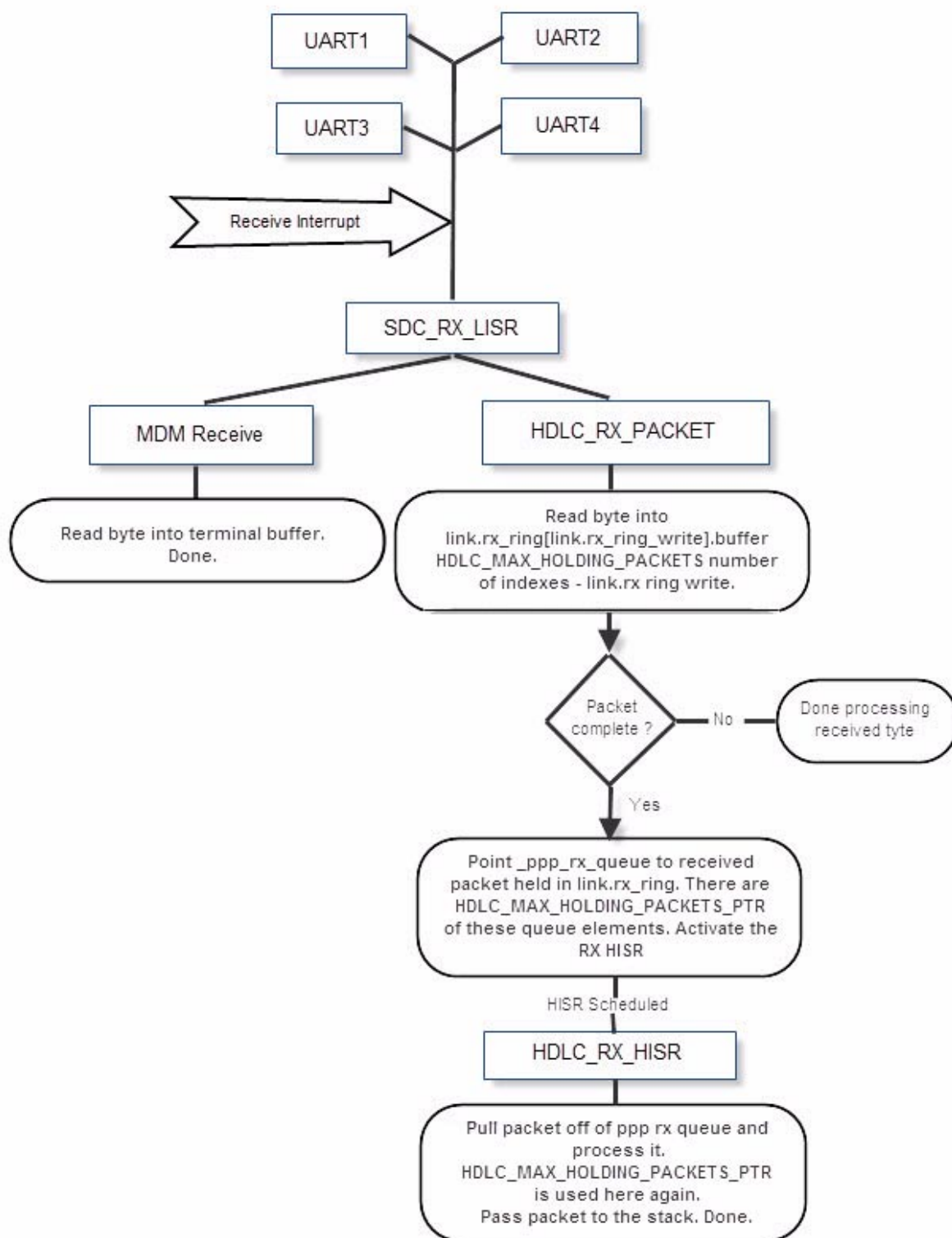
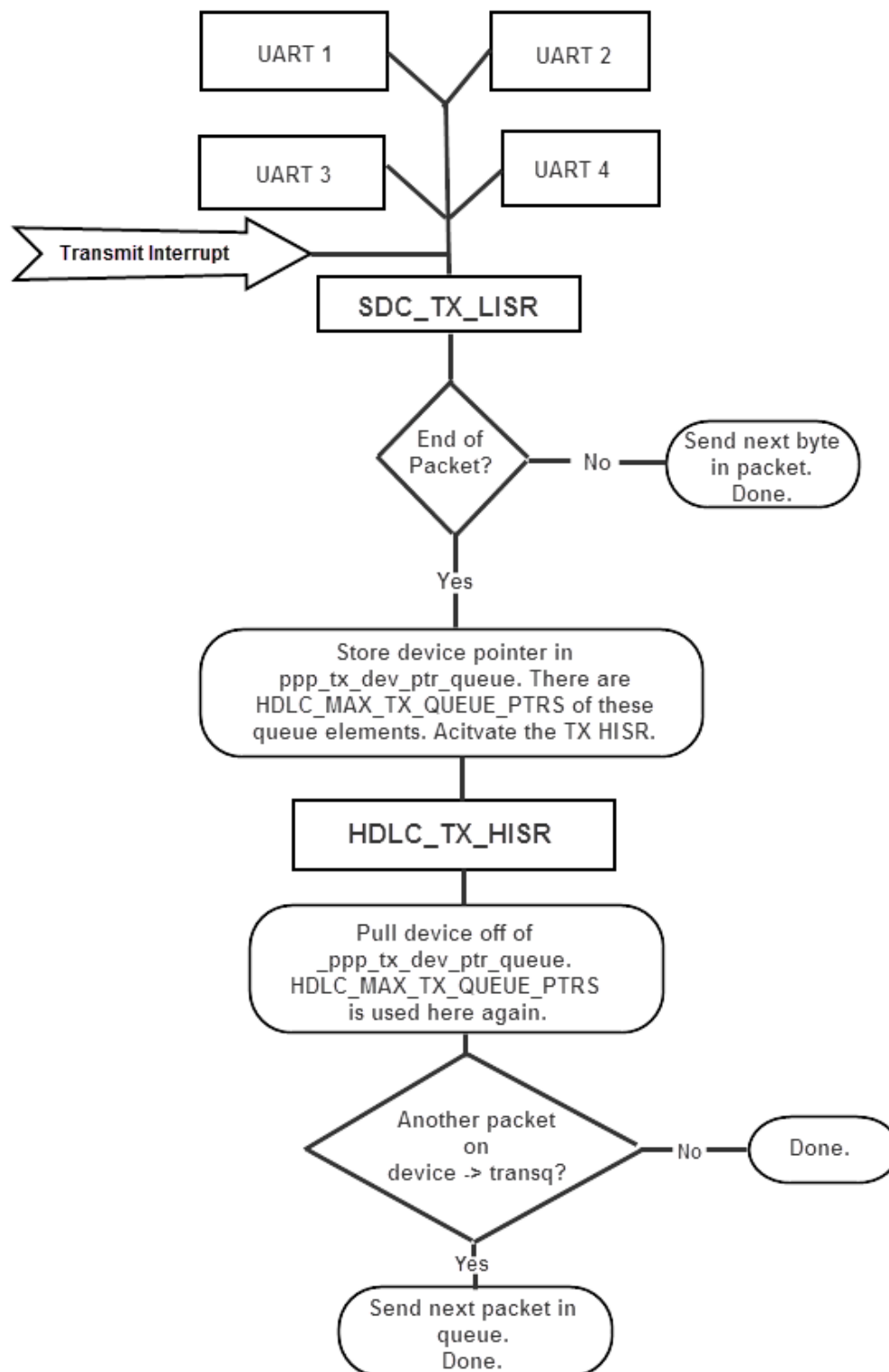




Figure 19-5. HDLC\_MAX\_TX\_QUEUE\_PTRS




## The Direct Cable Interface

Using the direct cable application, you can test PPP applications by connecting their target device to a standard PC via a null modem cable. The direct cable application was created and tested against a PC executing Windows and may require modification for communication with other operating systems. Also, note that depending on the hardware configuration of the target's UART a standard null modem cable may not work. Refer to the hardware specifications for the pin-out of the UART. A breakout box may be required to supply the correct pin configuration for the two ends of the link to be able to communicate.

Nucleus PPP3.2's Direct Cable Protocol can work for both PPP client and PPP server.

---

 **Note** The NULL Modem Protocol supported in Nucleus PPP version 3.1 and earlier, is no longer supported. It has been replaced by the Direct Cable Connection Protocol. These protocols are not compatible with each other.

---

## Setting up Direct Cable Connection for Windows

Perform the following steps to install a direct cable connection in Windows:

### Prerequisites

- A Windows machine (WIN-2k/WIN-XP)

### Procedure

1. Open the Control Panel
2. Select **Network and Dial-Up** Connections.
3. Double-click **Make New Connection**.
4. Select **Connect directly to another computer**.
5. Select **Guest** if Windows is acting as a PPP client, or select **Host** if Windows is acting as a PPP server.
6. Select the COM port on which the cable is connected.
7. Double-click the **Direct Connection** icon and enter a user name and password, and you will connect to the PPP server.

### Results

Once the connection is established, the two nodes communicate over PPP just as they did when using a modem.

**Note**

You have to wait to reconnect once you disconnect from the client from the server, because there are timers that need to time out before another connection can be made.

**Note**

The baud rates on which a direct cable (DC) application and Windows connect should be the same on both the peers.

## Related Topics

[The Direct Cable Interface](#)[Examples](#)

## Examples

An application note describing an example that demonstrates direct cable functionality for a PPP server and a PPP client in greater detail can be found on SupportNet:

<http://supportnet.mentor.com>

This is mainly used to aid in the development of a UART driver or to simplify application development and initial testing. By using the direct cable connection, you can avoid having to use a phone line and modem, thereby speeding the connection process during simple testing.

## DC Device Initialization

As mentioned briefly in the [PPP Overview](#), one of the ways that PPP network applications differ from network applications that use Ethernet or some other network medium is in the device initialization. A Nucleus PPP device is initialized differently as compared to the other devices in the Nucleus system. It uses an already initialized device (UART for example) and so it is just a matter of providing the correct information to the Nucleus NET [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) service.

By default, if PPP is enabled in the system, one of the PPP interfaces gets initialized based on the configurations enabled in the PPP *.metadata* file. To enable the DC interface, please ensure that the following options are configured as follows:

```
option("enable_serial") {
    default true
    description "This enables PPP communication over the serial interface."
}

option("dc_protocol_enable") {
    default true
    description "Enable the interface to use Direct Connect Cable Protocol"
}
```

## PPPoE Interface

This section describes the installation and configuration of the PPPoE module for Nucleus PPP executing on an embedded system.

## PPPoE Introduction

Nucleus PPPoE is an implementation of RFC 2516, which is a link layer protocol designed to allow multiple private point-to-point connections on an Ethernet network. PPPoE sits between the PPP layer and the driver for the physical hardware, an Ethernet driver. PPPoE allows multiple virtual PPP devices to be registered with Nucleus NET. Each of these devices can be used as either a PPP server or PPP client. To Nucleus NET these devices appear to be, and are treated as, true point-to-point connections. PPPoE is typically used for connecting a local device, such as a residential gateway, to a foreign PPPoE Access Concentrator (server) via a DSL modem.

Unlike the HDLC module, PPPoE does not include any API services. The PPPoE module does not interface directly to hardware. So there are no hardware dependencies for the PPPoE code.

## PPPoE Services

The main API for the PPP client/server routines are [NU\\_Dial\\_PPP\\_Server](#) and [NU\\_Wait\\_For\\_PPP\\_Client](#). These API calls are made in the same way for any PPP application using either serial or Ethernet interface, except for the following:

- [NU\\_Dial\\_PPP\\_Server](#)

When used with a serial UART for connecting to a peer through a telephone modem, the first parameter is a pointer to the telephone number of the PPP server. When used in a PPPoE link, it should contain a pointer to the service that is required of the Access Concentrator, as specified in RFC2516. This service is defined by a null terminated UINT8 string. All other parameters remain identical for both interfaces. See the description for [NU\\_Dial\\_PPP\\_Server](#) in the [Nucleus PPP Module](#) section for more information.

## PPPoE Device Initialization

As mentioned briefly in the [Nucleus PPP Driver Modules](#) section, one of the ways that PPP network applications differ from network applications that use Ethernet or some other network medium is in the device initialization. A Nucleus PPP device is initialized differently as compared to the other devices in the Nucleus system. It uses an already initialized device (UART for example) and so it is just a matter of providing the correct information to the Nucleus NET [NU\\_Init\\_Devices \(IPv4/IPv6\)](#) service.

By default, if PPP is enabled in the system, one of the PPP interfaces gets initialized based on the configurations enabled in the PPP *.metadata* file. To enable the PPPoE interface, please ensure that the following options are configured as follows:

```
option("enable_ethernet") {  
    default true  
    description "This enables PPP communication over the ethernet interface  
                also known as PPPoE."  
}
```

To use a specific ethernet port for PPPoE, please configure the following option as required:

```
option("eth_device") {  
    default      "eth0"  
    description "Name of the Ethernet device to be used for the PPPoE  
                communication"  
}
```

When a PPPoE device is registered with Nucleus NET, it is a virtual device. It does not interface directly to any hardware. This is unlike other devices such as UARTs and Ethernet, where there is a one-to-one match between hardware peripherals and device drivers that interface to that hardware. Each virtual PPPoE device ultimately uses a physical Ethernet device. In other words, instead of interfacing to hardware a PPPoE device interfaces to an Ethernet device that has been registered with Nucleus NET. It is important that the Ethernet device gets registered with Nucleus NET before registering a PPPoE device that makes use of that Ethernet device.

Any number of virtual devices may be linked to a single Ethernet device, but there are naturally limitations based on the memory and speed of your target hardware.

## PPPoE Configuration Options

Nucleus PPPoE was designed to be easily configurable. The files */drivers/ppp/src/ppec.c* and *include/drivers/ppe\_cfg.h* include constant definitions, data structures, and functions that can be modified by the application developer. Each of these is described in this section to provide information and guidance when customization is necessary.

## Macros and Definitions

### PPE\_AC\_NAME

Defines the name of the AC device in a null-terminated string. As shipped, this string is defined as “Mentor Graphics PPPoE Server” and needs to be set to a name of your choosing. This only applies if the system will act as an AC.

## PPE\_HOST\_NAME

Defines the name of the Host device in a null-terminated string. As shipped, this string is defined as “Mentor Graphics PPPoE Client” and needs to be set to a name of your choosing. This only applies if the system will act as a PPPoE Host (client).

## INCLUDE\_PPE\_AC

Define this preprocessor macro to `NU_TRUE` when compiling an application that will be used as an AC. If the system will never be used as an AC, then define to `NU_FALSE` to remove the applicable code from the build, thus reducing the code size. `NU_TRUE` is the default setting.

## INCLUDE\_PPE\_HOST

Define this preprocessor macro to `NU_TRUE` when compiling an application that will be used as a PPPoE host. If the system will never be used as a PPPoE host, then define to `NU_FALSE` to remove the applicable code from the build, thus reducing the code size. `NU_TRUE` is the default setting.

## PPE\_NUM\_SERVICES

This constant should contain the number of AC services that are defined in the `PPEC_AC_Services[]` list, which is configured in `/drivers/ppp/src/ppec.c`.

## Data Structures

### PPE\_SERVICE

`PPE_SERVICE` is the data structure for the AC services. This data structure is used for and would need to be modified when a service provider requires data that is specific to the service and needs to be provided as an argument to a function handler. The name of the service is required by the protocol. Other data may be added, but you would also have to add the code that uses this data.

```
typedef struct PPE_Services
{
    UINT8 *name;
    /*      UINT16 dummy1;      <-- Example -- not used by PPPoE! */
    /*      UINT16 dummy2;      <-- Example -- not used by PPPoE! */
} PPE_SERVICE;
```

## Globals

`/drivers/ppp/src/ppec.c` contains initialized data structures and functions that can be modified if necessary. Therefore, the application developer does not need to search through the other modules' code for any configurations to change.

## PPEC\_AC\_Services[]

Each AC service string that the system can handle is defined here. If there are other members in the service structure, then their values should be added here as well. Since this is in a C source file, this list can be defined using a process rather than by constants.

## PPEC\_Host\_Services[]

Similar to PPEC\_AC\_Services[], these services can be used by a host task when connecting to a foreign AC. This is not a requirement for your application, but is added merely for organizational convenience.

## PPEC\_CookieMask

Defines a mask that can be applied to the HOST-UNIQ tag or an AC Cookie tag. It is a null-terminated string, so one must be added to the string size to account for the terminator. This mask is applied to the given tag, as described in the following functions, to provide a level of security against improper use of the session information during discovery.

## PPEC Cookie Functions

The functions listed in this section describe the installation and configuration of the PPPoE module for Nucleus PPP executing on an embedded system.

- [PPEC\\_Encode\\_Cookie](#)
- [PPEC\\_Decode\\_Cookie](#)

## PPEC\_Encode\_Cookie

By default, this function applies a simple XOR of the mask previously described to the cookie tag. If greater security is needed, then you can modify this function accordingly.

### Usage

```
VOID PPEC_Encode_Cookie(UINT8  *macaddr,  
                        UINT16 sid,  
                        UINT8  *string);
```

### Arguments

- **macaddr**  
Ethernet address of the peer.
- **sid**  
An identifier for finding the correct virtual device.
- **string**  
Pointer to a pre-allocated memory buffer to store the results of this function.

### Example

```
/* The function variation provided below encodes the  
 * cookie using DES encryption instead of the less  
 * secure XOR operation.  
 *  
 * NOTE: The following example uses the OpenSSL APIs for  
 * DES encryption.  
 */  
VOID PPEC_Encode_Cookie(UINT8 *macaddr, UINT16 sid, UINT8 *string)  
{  
    const_DES_cblock    *key = "secretkey";  
    DES_key_schedule    schedule;  
    UINT8                *unenc_str;  
  
    /* Copy the ethernet address and "session id"  
     * to local string memory.  
     */  
    memcpy(unenc_str, macaddr, 6);  
    memcpy(&unenc_str[6], &sid, 2);  
  
    /* Set up the key-schedule */  
    DES_set_key((const_DES_cblock*)key, &schedule);  
  
    /* Make the request for encryption. */  
    DES_ecb_encrypt((const_DES_cblock*)unenc_str, (DES_cblock *)string,  
                    &schedule, DES_ENCRYPT);  
}
```



## Related Topics

[PPEC Cookie Functions](#)

## PPEC\_Decode\_Cookie

By default, this function applies a simple XOR of the mask previously described to the cookie tag. This effectively reverses the default encoding previously described. If the [PPEC\\_Encode\\_Cookie](#) function is modified to use a new encryption technique, then this function also needs to be modified to reverse the encryption.

### Usage

```
UINT16 PPEC_Decode_Cookie (PPE_TAG *tag);
```

### Arguments

- tag  
Pointer to the cookie tag.

### Return Values

- 0+  
The virtual device identifier.

### Example

```
/* The function variation provided below decodes the
 * cookie using DES decryption instead of the less
 * secure XOR operation.
 *
 * NOTE: The following example uses the OpenSSL APIs for
 * DES decryption.
 */
UINT16 PPEC_Decode_Cookie(PPE_TAG *tag)
{
    const_DES_cblock    *key = "secretkey";
    DES_key_schedule    schedule;
    UINT16              sid;
    UINT8               *unenc_str;

    /* Set up the key-schedule */
    DES_set_key((const_DES_cblock*)key, &schedule);

    /* Make the request for encryption. */
    DES_ecb_decrypt((const_DES_cblock*)tag->ppe_string, (DES_cblock *)
                    unenc_str, &schedule, DES_DECRYPT);

    /* The sid is always in native order. */
    memcpy(&sid, &unenc_str[6], 2);

    return sid;
}
```

### Related Topics

[PPEC Cookie Functions](#)

[PPEC\\_Encode\\_Cookie](#)

## Nucleus PPP Constants

This appendix contains all of the Nucleus PPP constants. Nucleus PPP services returns one of these constants or they return a Nucleus PLUS or Nucleus NET constant. Note that those constants in the NU\_PPE\_XXX form are specific to the Nucleus PPPoE module. [Table 19-8](#) lists all definitions created by Nucleus PPP:

### Nucleus PPP Constants (Alphabetical Listing)

**Table 19-8. Nucleus PPP Constants (Alphabetical Listing)**

Name	Decimal Value	Hex Value
NU_LCP_FAILED	-501	FFFFFE0B
NU_NCP_FAILED	-502	FFFFFE0A
NU_LOGIN_FAILED	-503	FFFFFE09
NU_NETWORK_BUSY	-504	FFFFFE08
NU_INVALID_LINK	-505	FFFFFE07
NU_NO_CONNECT	-506	FFFFFE06
NU_NO_CARRIER	-507	FFFFFE05
NU_BUSY	-508	FFFFFE04
NU_INVALID_MODE	-509	FFFFFE03
NU_NO_MODEM	-510	FFFFFE02
NU_PPP_ATTEMPT_ABORTED	-511	FFFFFE01
NU_PPP_INIT_FAILURE	-512	FFFFFE00
NU_NEGOTIATION_TIMEOUT	-513	FFFFFDFF
NU_NEGOTIATION_ABORTED	-514	FFFFFDFE
NU_PPP_INVALID_PROTOCOL	-515	FFFFFDFD
NU_AUTH_FAILED	-516	FFFFFDFC
NU_PPP_INVALID_PARAMS	-517	FFFFFDFB
NU_PPP_ENTRY_NOT_FOUND	-518	FFFFFDFA
NU_PPP_INVALID_OPTION	-519	FFFFFD F9
NU_INVALID_DEFAULT_MRU	-520	FFFFFD F8
NU_INVALID_DEFAULT_PROTOCOL	-521	FFFFFD F7
NU_INVALID_NUM_DNS_SERVERS	-522	FFFFFD F6

**Table 19-8. Nucleus PPP Constants (Alphabetical Listing) (cont.)**

Name	Decimal Value	Hex Value
NU_INVALID_PF_COMPRESSION	-523	FFFFFFDF5
NU_INVALID_AC_COMPRESSION	-524	FFFFFFDF4
NU_PPE_ERROR	-525	FFFFFFDF3
NU_PPE_PADx_ERROR	-526	FFFFFFDF2
NU_INVALID_REQUIRE_ENCRYPTION	-527	FFFFFFDF1
NU_INVALID_40BIT_ENCRYPTION	-528	FFFFFFDF0
NU_INVALID_56BIT_ENCRYPTION	-529	FFFFFFDEF
NU_INVALID_128BIT_ENCRYPTION	-530	FFFFFFDEE
NU_INVALID_USE_MAGIC_NUMBER	-531	FFFFFFDED
NU_INVALID_USE_ACCM	-532	FFFFFFDEC

## Additional Demonstrations

An application note describing several PPP demonstration applications in greater detail can be found on SupportNet:

<http://supportnet.mentor.com>









































































































# Appendix A

## Installing Certificates on Microsoft Windows

---

### Installing Certificates Procedure

This chapter contains the steps required to install the CA and user certificates on Microsoft® Windows XP. Setting up certificates on other versions of Microsoft Windows may be different than the steps shown here.

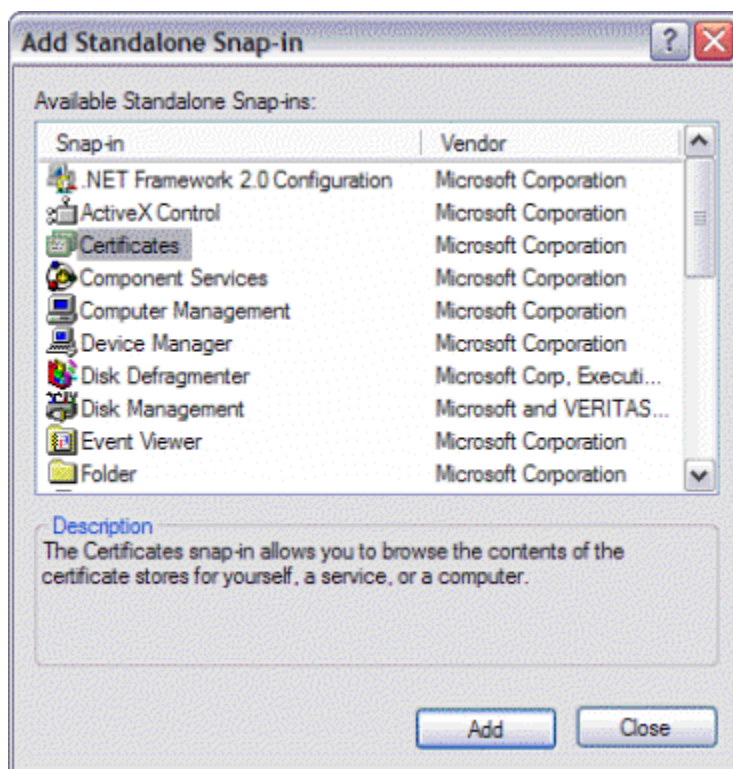
#### Prerequisites

- To import a certificate accompanied by a corresponding private key, both the certificate and the private key should be packed into a personal information exchange (PFX) file.

#### Procedure

To import the certificate, perform the following steps:

1. Select **Start > Run** and type **mmc**, then press **Enter**.
2. Click **File > Add/Remove Snap in** and then click the **Add...** button on the window that appears.
3. Select **Certificates** and click **Add**.

**Figure A-1. Add Standalone Snap-in**

4. On the wizard that starts, select **Computer Account** and click **Next**.
5. On the next screen select **Local Computer** and click **Finish**.
6. Click **Close** and then **OK** to close any open windows.
7. In the tree on the left, right-click on **Personal** to import a personal certificate, or **Trusted Root Certification Authorities** to import a CA's certificate and select **All Tasks > Import....**
8. A wizard appears. Browse to and select the certificate file (or the PFX file in case of personal certificate) and click **Next**.
9. If you are adding a certificate file, click **Next** and then click **Finish**.
10. If you are adding a personal certificate and private key in PFX format, after step 8, you are given a password prompt. Enter the password used when the PFX file was created, and then click **Next**. Complete the wizard as described in step 9.

## Related Topics

[Generating Certificates Using OpenSSL](#)

---

# Generating Certificates Using OpenSSL

OpenSSL is a free, open source library that supports manipulation of X.509 certificates. Certificates can easily be created and signed using command line tools provided with OpenSSL. For details on how to use these tools, see the documentation on the OpenSSL website (<http://www.openssl.org>).

## Procedure

This section describes how to create a certification authority and how to get certificates signed by that authority. Perform the following steps:

1. Install OpenSSL package on a Linux® or BSD® system. The same can be done in Cygwin® on Windows.
2. Create a new directory and copy the *openssl.cnf* file to this directory. This file is usually located at */etc/pki/tls/openssl.cnf*. Edit this file and set the directory paths listed in this file to convenient locations.
3. Copy the *CA.pl* script to the same directory mentioned in step 2. (This Perl script is installed with OpenSSL).
4. Run the following commands:

- a. Create new CA:

```
CA.pl -newca
```

- b. Create new certificate request:

```
CA.pl -newreq
```

- c. Sign the new certificate request with CA's certificate:

```
CA.pl -signreq
```

5. Under some environments (for example, Cygwin) this script needs to be invoked as follows:

```
perl -S CA.pl
```

## Related Topics

[Installing Certificates Procedure](#)



# Appendix B

## Installing Racoon2

---

### Installing Racoon2 Procedure

Racoon2 can be downloaded from the following:

<http://ftp.racoon2.wide.ad.jp/pub/racoon2/racoon2-current.tgz>

The file is a gzipped tar archive.

### Prerequisites

- Ensure that you have GCC installed.

### Procedure

Perform the following steps to extract the source, build and install Racoon2:

1. Run this command to extract the Racoon2 source:

```
$ tar -xzf racoon2-current.tgz
```

2. Run the following command to build and install:

```
$ make all install
```

3. Run the following commands to build and install:

```
$ make all install
```

### Results

Racoon2 is installed in the `/usr/local/racoon2` directory. The configuration files are placed in the `/usr/local/racoon2/etc` directory.

### Related Topics

[Racoon2 Configuration](#)

## Racoon2 Configuration

There are three basic configuration files and one topology specific configuration file for Racoon2. Configurations from all of these files can be added to a single file and the single file

be used. We have divided the configuration information into separate files to maintain clarity for this information.

The following are the three basic configuration files:

- `racoon2.conf`:

This file has information about port number used by racoon2 daemon. It also includes other configuration files.

- `vals.conf`:

This file contains information about where the pre-shared keys are placed and where the directory containing the digital certificates is. It also contains ID settings for peers, certificate and private key file names and IP addresses of peers. Scripts to run on certain actions can also be specified in this file.

- `default.conf`:

This file contains information about default values to be used with IKEv1 and IKEv2. Lifetime for SAs, resend intervals, algorithms to be used during the negotiation and a list of acceptable algorithms are all provided in this file. Some of the values in this file can be overridden in the topology specific configuration file.

The topology-specific configuration file can be either of *transport\_ike.conf* or *tunnel\_ike.conf*. These files have information about in what mode (tunnel or transport) IPsec works. These files also contain information on how to select packets for securing them with IPsec. Some of the values already defined in other configuration files can be overridden in these files.

## Required Configuration Changes for Use with Nucleus IPsec

### Changes Required in `racoon2.conf`

It is advisable to leave the original configuration files intact and make a copy of them, placing that copy in a convenient directory. When this has been done, *racoon2.conf* needs to be updated to reflect the path of other three include files. The following three lines need to be updated:

```
include "/usr/local/racoon2/etc/racoon2/vals.conf"
include "/usr/local/racoon2/etc/racoon2/default.conf"
include "/usr/local/racoon2/etc/racoon2/transport_ike.conf"
```

Replace the path in the quoted string to point to the configuration files that needs to be used.

Apart from this, only one change needs to be made. The line:

```
MY_IP port 500;
```



which is inside the `ike{ }` block inside `interface{ }` block needs to be changed to:

```
MY_IPADDRESS port 500;
```

## Changes Required in `vals.conf`

Line 5 must be changed to specify where the pre-shared key file is placed. This line defines a path variable named `PSKDIR`. Set the appropriate path in the quoted string.

If FQDN is to be used for authentication (which normally is not needed and authentication can easily be done using IP addresses only) set `MY_FQDN` and `PEER_FQDN` on lines 12 and 15 respectively to appropriate values. These values should match on both Nucleus and Racoon2 sides.

On lines 39 and 42, set `MY_IPADDRESS` and `PEER_IPADDRESS` to racoon2 node's and Nucleus node's IP addresses respectively. Note that `MY_IPADDRESS` here refers to the IP address of the system where Racoon2 is running.

## Changes Required in `default.conf`

The `ikev1{ }` and `kink{ }` blocks in this file are not used for IKEv2 and, hence, can be safely removed. If removed, set the first line in `remote{ }` block to:

```
acceptable_kmp{ikev2};
```

In the `ikev2{ }` block, set:

```
kmp_enc_alg to {3des_cbc}
```

```
kmp_prf_alg to {hmac_md5}
```

```
kmp_hash_alg to {hmac_md5}
```

```
kmp_dh_group to {modp1024}
```

Set the contents of the block starting with `"sa esp_01"` to the following:

```
sa_protocol esp;  
esp_enc_alg { 3des_cbc; };  
esp_auth_alg { hmac_md5; };
```

## Changes Required in `transport_ike.conf`

In the block `ikev2{ }`, change the following lines:

```
my_id fqdn "${MY_FQDN}";  
peers_id fqdn "${PEERS_FQDN}";
```

with:

```
my_id ipaddr "${MY_IPADDRESS}";  
peers_id ipaddr "${PEERS_IPADDRESS}";
```

## Racoon2 Daemon Usage

Racoon2 daemon can be started manually, or can be set to start automatically when the system boots.

The Racoon2 and SPMD daemons can be started using the following commands:

```
$ /usr/local/racoon2/sbin/spmd -d -f <path to racoon2.conf>  
$ /usr/local/racoon2/sbin/iked -d if <path to racoon2.conf>
```

Enter the following command to terminate both of these processes:

```
$ /usr/local/racoon2/sbin/spmd -k
```

# Embedded Software and Hardware License Agreement

The latest version of the Embedded Software and Hardware License Agreement is available on-line at:  
[www.mentor.com/eshla](http://www.mentor.com/eshla)

## IMPORTANT INFORMATION

**USE OF ALL PRODUCTS IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF PRODUCTS INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.**

## EMBEDDED SOFTWARE AND HARDWARE LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Products (as defined in Section 1) between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Products received electronically, certify destruction of Products and all accompanying items within five days after receipt of such Products and receive a full refund of any license fee paid.

1. **Definitions.** As used in this Agreement and any applicable quotation, supplement, attachment and/or addendum ("Addenda"), these terms shall have the following meanings:
  - 1.1. "Customer's Product" means Customer's end-user product identified by a unique SKU (including any Related SKUs) in an applicable Addenda that is developed, manufactured, branded and shipped solely by Customer or an authorized manufacturer or subcontractor on behalf of Customer to end-users or consumers;
  - 1.2. "Developer" means a unique user, as identified by a unique user identification number, with access to Embedded Software at an authorized Development Location. A unique user is an individual who works directly with the embedded software in source code form, or creates, modifies or compiles software that ultimately links to the Embedded Software in Object Code form and is embedded into Customer's Product at the point of manufacture;
  - 1.3. "Development Location" means the location where Products may be used as authorized in the applicable Addenda;
  - 1.4. "Development Tools" means the software that may be used by Customer for building, editing, compiling, debugging or prototyping Customer's Product;
  - 1.5. "Embedded Software" means Software that is embeddable;
  - 1.6. "End-User" means Customer's customer;
  - 1.7. "Executable Code" means a compiled program translated into a machine-readable format that can be loaded into memory and run by a certain processor;
  - 1.8. "Hardware" means a physically tangible electro-mechanical system or sub-system and associated documentation;
  - 1.9. "Linkable Object Code" or "Object Code" means linkable code resulting from the translation, processing, or compiling of Source Code by a computer into machine-readable format;
  - 1.10. "Mentor Embedded Linux" or "MEL" means Mentor Graphics' tools, source code, and recipes for building Linux systems;
  - 1.11. "Open Source Software" or "OSS" means software subject to an open source license which requires as a condition for redistribution of such software, including modifications thereto, that the: (i) redistribution be in source code form or be made available in source code form; (ii) redistributed software be licensed to allow the making of derivative works; or (iii) redistribution be at no charge;
  - 1.12. "Processor" means the specific microprocessor to be used with Software and implemented in Customer's Product;
  - 1.13. "Products" means Software, Term-Licensed Products and/or Hardware;
  - 1.14. "Proprietary Components" means the components of the Products that are owned and/or licensed by Mentor Graphics and are not subject to an Open Source Software license, as more fully set forth in the product documentation provided with the Products;

- 1.15. “Redistributable Components” means those components that are intended to be incorporated or linked into Customer’s Linkable Object Code developed with the Software, as more fully set forth in the documentation provided with the Products;
- 1.16. “Related SKU” means two or more Customer Products identified by logically-related SKUs, where there is no difference or change in the electrical hardware or software content between such Customer Products;
- 1.17. “Software” means software programs, Embedded Software and/or Development Tools, including any updates, modifications, revisions, copies, documentation and design data that are licensed under this Agreement;
- 1.18. “Source Code” means software in a form in which the program logic is readily understandable by a human being;
- 1.19. “Sourcery CodeBench Software” means Mentor Graphics’ Development Tool for C/C++ embedded application development;
- 1.20. “Sourcery VSIPL++” is Software providing C++ classes and functions for writing embedded signal processing applications designed to run on one or more processors;
- 1.21. “Stock Keeping Unit” or “SKU” is a unique number or code used to identify each distinct product, item or service available for purchase;
- 1.22. “Subsidiary” means any corporation more than 50% owned by Customer, excluding Mentor Graphics competitors. Customer agrees to fulfill the obligations of such Subsidiary in the event of default. To the extent Mentor Graphics authorizes any Subsidiary’s use of Products under this Agreement, Customer agrees to ensure such Subsidiary’s compliance with the terms of this Agreement and will be liable for any breach by a Subsidiary; and
- 1.23. “Term-Licensed Products” means Products licensed to Customer for a limited time period (“Term”).

## 2. Orders, Fees and Payment.

- 2.1. To the extent Customer (or if agreed by Mentor Graphics, Customer’s appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (“Order(s)”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement and any applicable Addenda, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 2.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. All invoices will be sent electronically to Customer on the date stated on the invoice unless otherwise specified in an Addendum. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer’s sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer’s behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 2.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics’ delivery of Software by electronic means is subject to Customer’s provision of both a primary and an alternate e-mail address.

## 3. Grant of License.

- 3.1. The Products installed, downloaded, or otherwise acquired by Customer under this Agreement constitute or contain copyrighted, trade secret, proprietary and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software as described in the applicable Addenda. The limited licenses granted under the applicable Addenda shall continue until the expiration date of Term-Licensed Products or termination in accordance with Section 12 below, whichever occurs first. **Mentor Graphics does NOT grant Customer any right to (a) sublicense or (b) use Software beyond the scope of this Section without first signing a separate agreement or Addenda with Mentor Graphics for such purpose.**
- 3.2. License Type. The license type shall be identified in the applicable Addenda.
- 3.2.1. Development License: During the Term, if any, Customer may modify, compile, assemble and convert the applicable Embedded Software Source Code into Linkable Object Code and/or Executable Code form by the number of Developers specified, for the Processor(s), Customer’s Product(s) and at the Development Location(s) identified in the applicable Addenda.

- 3.2.2. End-User Product License: During the Term, if any, and unless otherwise specified in the applicable Addenda, Customer may incorporate or embed an Executable Code version of the Embedded Software into the specified number of copies of Customer's Product(s), using the Processor Unit(s), and at the Development Location(s) identified in the applicable Addenda. Customer may manufacture, brand and distribute such Customer's Product(s) worldwide to its End-Users.
- 3.2.3. Internal Tool License: During the Term, if any, Customer may use the Development Tools solely: (a) for internal business purposes and (b) on the specified number of computer work stations and sites. Development Tools are licensed on a per-seat or floating basis, as specified in the applicable Addenda, and shall not be distributed to others or delivered in Customer's Product(s) unless specifically authorized in an applicable Addenda.
- 3.2.4. Sourcery CodeBench Professional Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software (i) if the license is a node-locked license, by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, or (ii) if the license is a floating license, by the authorized number of concurrent users on one or more machines provided that only the authorized number of copies of the Software are in use at any one time, and (b) distribute the Redistributable Components of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.5. Sourcery CodeBench Standard Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on up to two machines provided that only one copy of the Software is in use at any one time, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer's Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.6. Sourcery CodeBench Personal Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.2.7. Sourcery CodeBench Academic Edition License: During the Term specified in the applicable Addenda, Customer may (a) install and use the Proprietary Components of the Software for non-commercial, academic purposes only by a single user who uses the Software on one machine, and (b) distribute the Redistributable Component(s) of the Software in Executable Code form only and only as part of Customer Object Code developed with the Software that provides substantially different functionality than the Redistributable Component(s) alone.
- 3.3. Mentor Graphics may from time to time, in its sole discretion, lend Products to Customer. For each loan, Mentor Graphics will identify in writing the quantity and description of Software loaned, the authorized location and the Term of the loan. Mentor Graphics will grant to Customer a temporary license to use the loaned Software solely for Customer's internal evaluation in a non-production environment. Customer shall return to Mentor Graphics or delete and destroy loaned Software on or before the expiration of the loan Term. Customer will sign a certification of such deletion or destruction if requested by Mentor Graphics.

#### **4. Beta Code.**

- 4.1. Portions or all of certain Products may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
- 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
- 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

## 5. Restrictions on Use.

- 5.1. Customer may copy Software only as reasonably necessary to support the authorized use, including archival and backup purposes. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Except where embedded in Executable Code form in Customer's Product, Customer shall maintain a record of the number and location of all copies of Software, including copies merged with other software and products, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees, authorized manufacturers or authorized contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics immediate written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use.
- 5.2. Customer acknowledges that the Products provided hereunder may contain Source Code which is proprietary and its confidentiality is of the highest importance and value to Mentor Graphics. Customer acknowledges that Mentor Graphics may be seriously harmed if such Source Code is disclosed in violation of this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any Source Code from Products that are not provided in Source Code form. Except as embedded in Executable Code in Customer's Product and distributed in the ordinary course of business, in no event shall Customer provide Products to Mentor Graphics competitors. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Under no circumstances shall Customer use Products or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.
- 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent, which shall not be unreasonably withheld, and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
- 5.4. Notwithstanding any provision in an OSS license agreement applicable to a component of the Sourcery CodeBench Software that permits the redistribution of such component to a third party in Source Code or binary form, Customer may not use any Mentor Graphics trademark, whether registered or unregistered, in connection with such distribution, and may not recompile the Open Source Software components with the --with-pkgversion or --with-bugurl configuration options that embed Mentor Graphics' trademarks in the resulting binary.
- 5.5. The provisions of this Section 5 shall survive the termination of this Agreement.

## 6. Support Services.

- 6.1. Except as described in Sections 6.2, 6.3 and 6.4 below, and unless otherwise specified in any applicable Addenda to this Agreement, to the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current End-User Software Support Terms located at <http://supportnet.mentor.com/about/legal/>.
- 6.2. To the extent Customer purchases support services for Sourcery CodeBench Software, support will be provided solely in accordance with the provisions of this Section 6.2. Mentor Graphics shall provide updates and technical support to Customer as described herein only on the condition that Customer uses the Executable Code form of the Sourcery CodeBench Software for internal use only and/or distributes the Redistributable Components in Executable Code form only (except as provided in a separate redistribution agreement with Mentor Graphics or as required by the applicable Open Source license). Any other distribution by Customer of the Sourcery CodeBench Software (or any component thereof) in any form, including distribution permitted by the applicable Open Source license, shall automatically terminate any remaining support term. Subject to the foregoing and the payment of support fees, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased in accordance with Mentor Graphics' then-current Sourcery CodeBench Software Support Terms located at <http://www.mentor.com/codebench-support-legal>.
- 6.3. To the extent Customer purchases support services for Sourcery VSIPL++, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased solely in accordance with Mentor Graphics' then-current Sourcery VSIPL++ Support Terms located at <http://www.mentor.com/vsipl-support-legal>.
- 6.4. To the extent Customer purchases support services for Mentor Embedded Linux, Mentor Graphics will provide Customer updates and technical support for the number of Developers at the Development Location(s) for which support is purchased



solely in accordance with Mentor Graphics' then-current Mentor Embedded Linux Support Terms located at <http://www.mentor.com/mel-support-legal>.

7. **Third Party and Open Source Software.** Products may contain Open Source Software or code distributed under a proprietary third party license agreement. Please see applicable Products documentation, including but not limited to license notice files, header files or source code for further details. Please see the applicable Open Source Software license(s) for additional rights and obligations governing your use and distribution of Open Source Software. Customer agrees that it shall not subject any Product provided by Mentor Graphics under this Agreement to any Open Source Software license that does not otherwise apply to such Product. In the event of conflict between the terms of this Agreement, any Addenda and an applicable OSS or proprietary third party agreement, the OSS or proprietary third party agreement will control solely with respect to the OSS or proprietary third party software component. The provisions of this Section 7 shall survive the termination of this Agreement.
8. **Limited Warranty.**
  - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual and/or specification. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Products under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; OR (B) PRODUCTS PROVIDED AT NO CHARGE, WHICH ARE PROVIDED "AS IS" UNLESS OTHERWISE AGREED IN WRITING.
  - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE TO CUSTOMER AND DO NOT APPLY TO ANY END-USER. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, WITH RESPECT TO PRODUCTS OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, AND EXCEPT FOR EITHER PARTY'S BREACH OF ITS CONFIDENTIALITY OBLIGATIONS, CUSTOMER'S BREACH OF LICENSING TERMS OR CUSTOMER'S OBLIGATIONS UNDER SECTION 10, IN NO EVENT SHALL: (A) EITHER PARTY OR ITS RESPECTIVE LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF SUCH PARTY OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES; AND (B) EITHER PARTY OR ITS RESPECTIVE LICENSORS' LIABILITY UNDER THIS AGREEMENT, INCLUDING, FOR THE AVOIDANCE OF DOUBT, LIABILITY FOR ATTORNEYS' FEES OR COSTS, EXCEED THE GREATER OF THE FEES PAID OR OWING TO MENTOR GRAPHICS FOR THE PRODUCT OR SERVICE GIVING RISE TO THE CLAIM OR \$500,000 (FIVE HUNDRED THOUSAND U.S. DOLLARS). NOTWITHSTANDING THE FOREGOING, IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **Hazardous Applications.**
  - 10.1. Customer agrees that Mentor Graphics has no control over Customer's testing or the specific applications and use that Customer will make of Products. Mentor Graphics Products are not specifically designed for use in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support systems, medical devices or other applications in which the failure of Mentor Graphics Products could lead to death, personal injury, or severe physical or environmental damage ("Hazardous Applications").
  - 10.2. CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING PRODUCTS USED IN HAZARDOUS APPLICATIONS AND SHALL BE SOLELY LIABLE FOR ANY DAMAGES RESULTING FROM SUCH USE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF PRODUCTS IN ANY HAZARDOUS APPLICATIONS.
  - 10.3. CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING REASONABLE ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10.1.
  - 10.4. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

## 11. Infringement.

- 11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.
  - 11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, and in addition to its obligations under Section 11.1, either (a) replace or modify the Product so that it becomes noninfringing; or (b) procure for Customer the right to continue using the Product. If Mentor Graphics determines that neither of those alternatives is financially practical or otherwise reasonably available, Mentor Graphics may require the return of the Product and refund to Customer any purchase price or license fee(s) paid.
  - 11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of the Product with any product not furnished by Mentor Graphics, where the Product itself is not infringing; (b) the modification of the Product other than by Mentor Graphics or as directed by Mentor Graphics, where the unmodified Product would not infringe; (c) the use of the infringing Product when Mentor Graphics has provided Customer with a current unaltered release of a non-infringing Product of substantially similar functionality in accordance with Subsection 11.2(a); (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells, where the Product itself is not infringing; (f) any Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; (h) Open Source Software, except to the extent that the infringement is directly caused by Mentor Graphics' modifications to such Open Source Software; or (i) infringement by Customer that is deemed willful. In the case of (i), Customer shall reimburse Mentor Graphics for its reasonable attorneys' fees and other costs related to the action.
  - 11.4. THIS SECTION 11 IS SUBJECT TO SECTION 9 ABOVE AND STATES: (A) THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND (B) CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
12. **Termination and Effect of Termination.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized Term.
- 12.1. Termination for Breach. This Agreement shall remain in effect until terminated in accordance with its terms. Mentor Graphics may terminate this Agreement and/or any licenses granted under this Agreement, and Customer will immediately discontinue use and distribution of Products, if Customer (a) commits any material breach of any provision of this Agreement and fails to cure such breach upon 30-days prior written notice; or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination. For the avoidance of doubt, nothing in this Section 12 shall be construed to prevent Mentor Graphics from seeking immediate injunctive relief in the event of any threatened or actual breach of Customer's obligations hereunder.
  - 12.2. Effect of Termination. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination or expiration of the Term, Customer will discontinue use and/or distribution of Products, and shall return Hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form, except to the extent an Open Source Software license conflicts with this Section 12.2 and permits Customer's continued use of any Open Source Software portion or component of a Product. Upon termination for Customer's breach, an End-User may continue its use and/or distribution of Customer's Product so long as: (a) the End-User was licensed according to the terms of this Agreement, if applicable to such End-User, and (b) such End-User is not in breach of its agreement, if applicable, nor a party to Customer's breach.
13. **Export.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies. Customer acknowledges that the regulation of product export is in continuous modification by local governments and/or the United States Congress and administrative agencies. Customer agrees to complete all documents and to meet all requirements arising out of such modifications.
14. **U.S. Government License Rights.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.



15. **Third Party Beneficiary.** For any Products licensed under this Agreement and provided by Customer to End-Users, Mentor Graphics or the applicable licensor is a third party beneficiary of the agreement between Customer and End-User. Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
16. **Review of License Usage.** Customer will monitor the access to and use of Software. With prior written notice, during Customer's normal business hours, and no more frequently than once per calendar year, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system, records, accounts and sublicensing documents deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all Customer information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. Such license review shall be at Mentor Graphics' expense unless it reveals a material underpayment of fees of five percent or more, in which case Customer shall reimburse Mentor Graphics for the costs of such license review. Customer shall promptly pay any such fees. If the license review reveals that Customer has made an overpayment, Mentor Graphics has the option to either provide the Customer with a refund or credit the amount overpaid to Customer's next payment. The provisions of this Section 16 shall survive the termination of this Agreement.
17. **Controlling Law, Jurisdiction and Dispute Resolution.** This Agreement shall be governed by and construed under the laws of the State of California, USA, excluding choice of law rules. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the state and federal courts of California, USA. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer or its Subsidiary in the jurisdiction where Customer's or its Subsidiary's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
18. **Severability.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
19. **Miscellaneous.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.